RICE UNIVERSITY

Multi-robot Behaviors with Bearing-only Sensors and Scale-free Coordinates

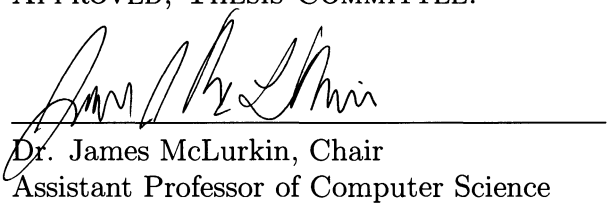
by

Andrew J. Lynch

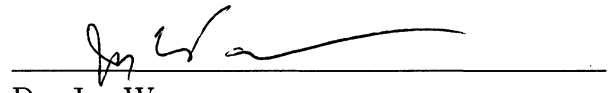
A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Master of Science

Approved, Thesis Committee:



Dr. James McLurkin, Chair
Assistant Professor of Computer Science



Dr. Joe Warren
Professor of Computer Science

Dr. Marcia K. O'Malley
Associate Professor of Mechanical
Engineering and Materials Science

Houston, Texas

December, 2011

Abstract

Multi-robot Behaviors with Bearing-only Sensors and Scale-free Coordinates

by

Andrew J. Lynch

This thesis presents a low-cost multi-robot system for large populations of robots, a new coordinate system for the robot based on angles between robots and a series of experiments validating robot performance. The new robot platform, the *r-one* will serve as an educational, outreach and research platform for robotics. I consider the robot's *bearing-only* sensor model, where each robot is capable of measuring the bearing, but not the distance, to each of its neighbors. This work also includes behaviors demonstrating the efficiency of this approach with this bearing-only sensor model. The new local coordinate systems based on angular information is introduced as *scale-free coordinate system.* Each robot produces its own local scale-free coordinates to determine the relative positions of its neighbors up to an unknown scaling factor. The computation of scale-free coordinates is analyzed with hardware and simulation validation. For hardware, the scale-free algorithm is tailored to low-cost systems with limited communication bandwidth and sensor resolution. The algorithm also uses a noise sensitivity model to reduce the impact of noise on the computed scale-free coordinates. I validate the algorithm with static and dynamic motion experiments.

Acknowledgments

I dedicate this work to my family and friends. Their patience has made my experience with Rice possible. Frances Liao has been my driving force on motivation and dedication to my work.

I also express many thanks to my advisor, Professor James McLurkin. His guidance and direction has been a strong influence in my writing and research. In addition, Dr. Rixner, Dr. Warren and Dr. O'Malley have been a tremendous resource in their guidance on writing and presentation.

Many thanks to my fellow graduate students, Alejandro Cornejo and Devin Grady. Mr. Cornejo has been a close collaborator on the entirety of the scale-free coordinate work. In addition, Rice undergraduates Elizabeth Fudge and Siegfried Bielstein have contributed immensely to developing scale-free coordinates.

I also want to thank the Rice technicians, Joe Gesenhues and Carlos Amaro for their assistance in fabrication and laser cutting of parts.

The summer design periods were the most productive construction phases of the robots and would not be possible without the army of undergraduates and high-school interns. In summer 2010, Thomas Barr, Siegfried Bilstein, Kathleen Foster, Alvin Chou, Lingo Dai, Lan Li, Joan Chao , Dennis Qian, Brian Shields, James Hill, Brandon Heath, Chris Licato, Nelson Chen and Joshua Bryant all contributed to finalize the r-one V6 design. In summer 2011, Chris Licato, Nelson Chen, Joshua Lipschultz, Sunny Kim, Meagan John, Jen Shen, William Li, Nnenna Okeke, Nathan Alison, Ebrima Jobe, Tarik Ward, Jennifer Shen all contributed to the r-one V11 design.

The Rice computer science support staff has also been a huge assistance in writing, procedures and guidelines. Thank you to Beth Rivera for putting up with our insane amount of orders and paperwork. Thank you to Bel Martinez in helping shepard my graduate experience with computer science to completion.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The acceleration of embedded computation and sensing is creating a bright future for low-cost *multi-robot* systems. While multi-robot systems have been studied for over 22 years [1], the feasibility and engineering hurdles have presented major barriers in the field. In the past, single robot applications have dominated the media and history of robotics. For example, single robots have shown commercial success in robot vacuuming [2], drone surveillance [3] and autonomous driving [4]. How do these applications apply in the multi-robot context ? Let's take an example, imagine cleaning your house with a single robot vacuum or performing surveillance of a city with one aerial robot. Now imagine cleaning your house with a team of robot vacuums or performing surveillance with a swarm of robots much more efficiently. The key idea is how the robots work together to accomplish the task compared to a group of single robots all operating independently. Therefore, multiple robots is the next logical step in robotics because a single robot vacuuming a large building will never be as effective as a team of robot vacuums cooperating together. Despite the success of single robot applications, multi-robot systems are still filled with challenges in hardware and software. Multi-robot distributed sensing, actuation, communication and control need integration into a standardized system.

1.1 Contributions

This work introduces a new multi-robot platform, a novel coordinate system for the platform and a series of control experiments to validate the performance. The brief terminology overview of each system will be mentioned in the introduction followed by an in-depth

(a) r-one. (b) r-ones in configuration. (c) multi-robot cost.

Figure 1.1: **a:** The r-one robot was designed by MRSL group at Rice University for multi-robot research. **b:** Multiple r-one robots for testing procedure. **c:** The four common multi-robot platforms and corresponding cost comparison. Each robot has a different capability set. See discussion in the second chapter for more details about the related multi-robot designs.

discussion in the subsequent chapters.

1.1.1 The r-one: Multi-Robot System

The new multi-robot platform developed in this work is called the *r-one* robot as shown in Figure 1.1(a). The *r-one* is a multi-robot system designed for research, education and outreach. The robot design has struck a balance between cost and functionality at a price of $245 per robot. The affordable cost of the robot allows scaling to large populations of robots such as 30 or more robots as shown in Figure 1.1(b). This cost problem has not been solved effectively in multi-robot systems [5]. The r-one platform enables multi-robot research at a practical cost as compared with the prices of the other available platforms as shown in Figure 1.1(c). The closest competitors have similar multi-robot capabilities but are prohibitively expensive for education and research purposes.

In terms of education and outreach, the r-one platform delivers a high-quality robot experience for new students learning how to program. To welcome new programmers to the platform, a fully embedded python interpreter has been created for the r-one by Rixner and Barr [5]. The python environment was used successfully in two years of

freshmen engineering introduction course. The python environment on embedded system with sensors and motors offers an interactive learning opportunity for students.

1.1.2 Multi-Robot Coordinate Systems

The r-one found a balance between cost and functionality by using *angle-only* sensors for localization and communication. Angle-only means quite literally the robot only has the bearing and orientation of neighboring robots. The *orientation* of the neighbor is the angle between the neighbor's heading with respect to the sensing robot. To address the lack of range in angle-only coordinate systems, this work has introduced a new coordinate system. Figure 1.2 illustrates the concept of measuring bearing and not range between a neighbor robot.

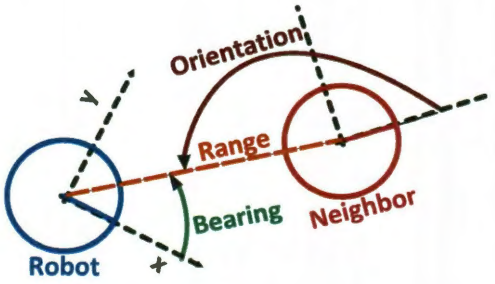


Figure 1.2: A model of bearing range between robots. The sensing robot on the left is centered at the origin (0,0) and x-axis is centered with the *heading* of the robot. Range is unknown for the bearing-only coordinate systems. The *orientation* of the neighbor is the angle between the neighbor's heading with respect to the sensing robot x-axis.

To obtain range in our new coordinate system, the robots must communicate with other robots in a network. Multiple robots in a network need the ability to measure the geometry of the communication network graph also defined as the *network geometry*. Geometric information of the network is necessary in configuration control algorithms with relative movements between robots.

A *global positioning system* or GPS system can provide geometric information of the network such as in Batalin [6]. However, indoor environments do not have access to GPS

and therefore require another type of local sensing. For large populations of robots, there exists a need for low-cost and simple sensors to obtain the network geometry.

Local Network Geometry. This work is focused on estimating *local network geometry*: measuring the pose (x, y, θ) of neighbors from a robot's local reference frame [7]. Figure 1.3 illustrates the local network geometry of the communication formed by the network. The triangles formed between robots are based on bearing angle measurements shared between the network.

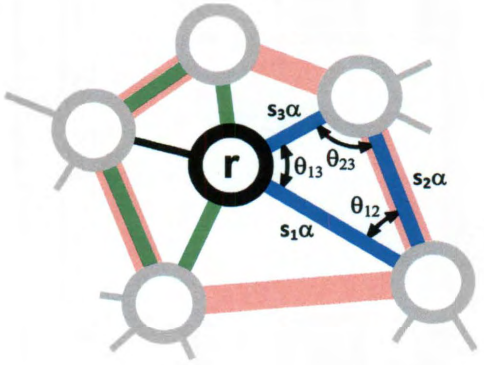


Figure 1.3: Local network geometry with inter-angles between neighbors form the communication geometry.

There are many possible sensor approaches to computing local network geometry with trade-offs between cost, complexity and communication requirements [8, 9, 10]. Directly measuring pose (x, y, θ) of neighboring robots is the most powerful type of sensing. Determining a neighbor pose requires bearing and range measurements of a neighboring robot. Measuring only bearing or range reduces the cost of the sensing on the robot.

The r-one robots take advantage of bearing-only sensors to measure local network geometry. With only angular information, a robot has the capability to execute a clustering, sorted order following and a number of other algorithms [11, 12, 13]. However, this information is insufficient to directly compute all the parameters of its local network geometry.

Scale-Free Vornoi Example.

As an example, consider the canonical problem of controlling a multi-robot system to a centroidal Voronoi configuration [14]. This is straightforward to solve with the full pose of neighboring robots. However, with only bearing measurements, the problem becomes more difficult. For example, Figure 1.4 illustrates two configurations with equivalent bearing measurements but very different Voronoi cells.

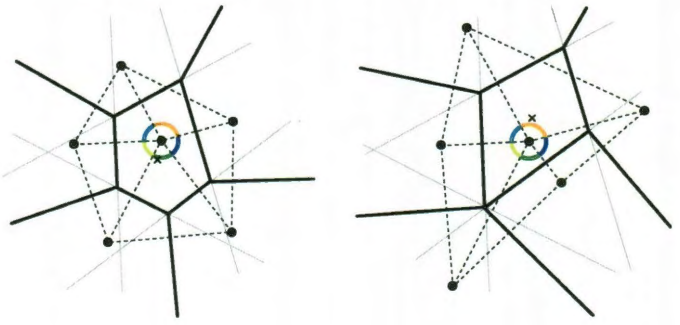


Figure 1.4: Two distinct Voronoi cells with the same bearing measurements.

This work discusses the concept of local *scale-free coordinates*. This allows each robot to use bearing measurements from its local network to determine the relative positions of its neighbors up to an unknown scaling factor, α, such that pose of a neighbor is given by $p = \{\alpha x, \alpha y, \theta\}$. Returning to the vornoi cell example, scale-free coordinates are sufficient to compute the centroid of a robot's Voronoi cell. However, since scale-free distances have no units, the robot cannot distinguish between 3 m or 3 cm distance to the centroid. This presents challenges to algorithms, in particular to motion control, which we consider in the experiment in Section 6.2. This paper argues that local scale-free coordinates are slightly weaker than complete local coordinates, but more practical than only bearing measurements. Scale-free coordinates relies on communication bandwidth to extract more positioning information from the network compared to bearing-only coordinate systems.

1.1.3 Experimental Validation

To verify the new multi-robot platform and the novel coordinate system, I will present a series of validation experiments. These experiments involve a robot moving to a centroid of a shape of robots using scale-free coordinates. In addition, a series of multi-robot behaviors such as clustering, flocking and sorted-order following will be presented to showcase performance of the system. These behaviors are interesting because they rely only on bearing measurements compared to the standard approach with range-bearing information.

1.2 Summary

This work consists of introducing a new robot platform, a novel coordinate system along with robot experiments to validate the platform. The second chapter outlines the related work of multi-robot platforms, coordinates systems and multi-robot experiments. The third chapter dives into the hardware details of the new multi-robot platform. The fourth chapter presents the model of robot network and coordinate system constraints. The fifth chapter investigates the computation of a robot's local scale-free coordinates in random configurations using different degree and communication parameters.

To conclude the work, chapter six also presents a series of hardware experiments and robot behaviors.

Chapter 2

Related Work

This chapter overviews related approaches in multi-robot systems and *angular* coordinates. The goal of this chapter is to introduce the key features of a multi-robot system and show how the robotics community has progressed over the years. A more in-depth related work section is also provided for each chapter to help distribute the information for the reader.

2.1 Distributed Multi-robot Attributes

Fully distributed multi-robot systems are designed for distributed sensing, actuation, communication and control as shown in Figure 2.1. Distributed local sensing of obstacles and neighboring robots sets the foundation of multi-robot systems. The robotics community lacks a readily available sensor package for directional communication and obstacle detection for small robots [5]. The majority of fully distributed multi-robot systems build their own unique approach to distributed local sensing.

There exist many *partial* multi-robot platforms due to their lack of distributed sensing, actuation, communication or control. Many of these systems have artificially created communication or sensing through a centralized server relaying information to the robots. The centralized server approach is suitable for validating distributed actuation and control tasks, as presented in Das [15]. Distributed actuation consists of each robot being equipped with mobility such as motors for movement. Executing a closed loop controller on each robot extends actuation to distributed control.

This work is interested in fully distributed behaviors with physical hardware. To effectively execute distributed behaviors, multi-robot systems need distributed communication

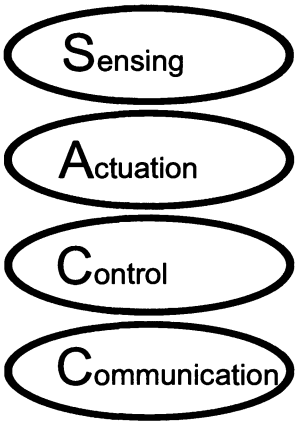
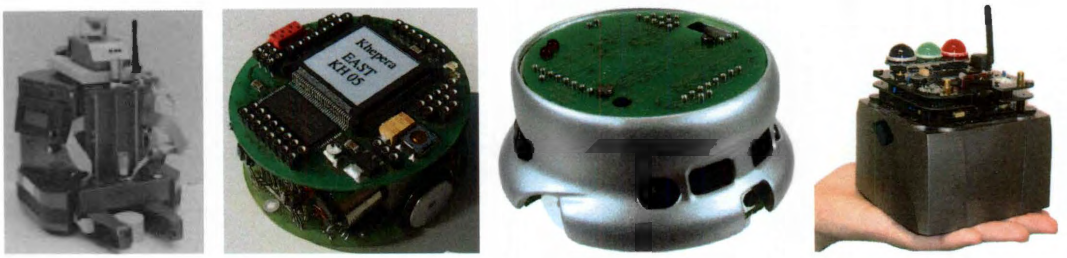


Figure 2.1: The four components of fully distributed multi-robot systems. Distributed sensing has local sensing on each robot. Distributed actuation is the ability for each robot to move. Distributed control combines together on-board sensing and actuation into local control systems on each robot. Distributed communication uses line-of-sight communication between neighboring robots.

with other robots to exchange information about the network geometry. Distributed communication can be achieved with line-of-sight directional communication sensors such as infrared receivers and transmitters. Communicating with an omni-directional antenna system does not scale for large distributed systems due to bandwidth constraints. Thus, line-of-sight communication is the preferred method of distributed communication.



(a) Mataric R2. (b) Khepera. (c) Khepera II. (d) SwarmBot.

Figure 2.2: **a:** The MIT R2 robot built in 1994 [16]. **b:** The Khepera built in 1996 [17].
c: The Khepera II built in 1999 as a multi-robot platform.
d: The SwarmBot built in 2003 as a fully distributed multi-robot platform [19].

2.2 Early Multi-Robot Systems

One of the original multi-robot systems began with the Mataric platform R2 platform in 1994, Figure 2.2(a) [16]. The R2 operated in groups of four to achieve group behaviors such as foraging and flocking. These robots were a significant milestone in the field but lacked the communication and sensing features for a fully distributed multi-robot. The robotics community took another step in multi-robot systems to reduce the physical size of the robot. EPFL from Switzerland introduced one of the first implementations of a physically small multi-robot system by upgrading the original Khepera robot created in 1996 with distributed control to create the Khepera II in 1999 [17] as shown in Figure 2.2(c). Another significant leap in multi-robot systems was the design of a fully distributed multi-robot system called the SwarmBot developed by McLurkin and iRobot [18] as shown in Figure 3.2(c). The SwarmBot featured bump skirts, infrared, tracking beacons and on-board cameras. The SwarmBots now reside at Rice University for research experiments. The r-one robot presented in this work has been designed with lessons learned from the SwarmBots.

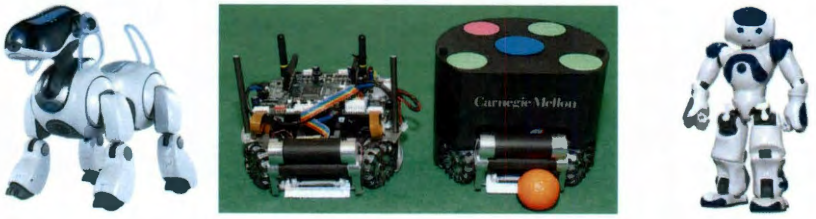
(a) Auto Assembly [22].

(b) Kiva [23].

Figure 2.3: **a:** Auto Assembly line with Kuka Robots performing distributed actuation. These are controlled by a centralized computer, thus not fully distributed systems. **b:** Kiva package distribution system are a fully distributed mobile multi-robot.

Multi-robot Systems in Industry.

Commercial development of multi-robot technologies have also progressed with specific industries. With a proper distributed platform and a simple task, large populations of robots scale successfully for package movement or assembly line manufacturing. For example, the auto industry has combined multiple robots together for assembly of vehicles for many years now [20] as shown in Figure 2.3(a) However this system is not fully distributed and typically has a centralized server relaying actuation commands to each robot. More recently, Kiva systems is the first commercial example of a fully distributed multi-robot system for warehouse package distribution [21] as shown in Figure 2.3(b). Automated package routing methods have been in the industry for a long period of time such as AS/RS and carousel conveyor belts. However, Kiva Systems' early customers were not satisfied with the available automated multi-carousel systems for package management. These traditional warehouse distribution carousels are expensive and under-utilized until volume reaches the carousel capacity. In contrast, multi-robot systems scale by adjusting the number of robots and shelves depending on volume [23].These first commercial applications in the automobile assembly and warehouse distribution have grown rapidly indicating that multi-robot systems provide tremendous potential to specific industries.



(a) AIBO [24]. (b) RoboCup [25]. (c) NAO [26].

Figure 2.4: **a:** The Sony AIBO robot dog used in the RoboCup competition.
b: The CMDragons small-size RoboCup team from Carnegie Mellon.
c: The Aldebaran NAO Humanoid robot used in the RoboCup competition.

Multi-robot Research Competitions.

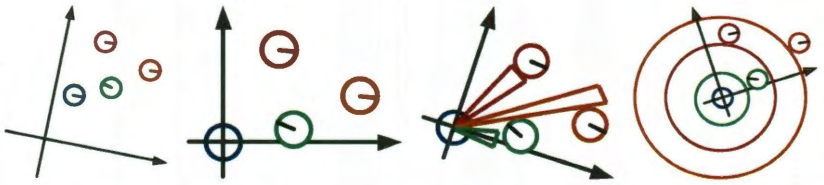
To develop this potential from research, the robotics landscape is shifting to address multiple robot challenges such as mapping [12], exploration [27], search-and-rescue [28], and surveillance [29]. Specifically, the research community has introduced specific multi-robot competitions such as robotic soccer: RoboCup [30] and military surveillance: MAGIC competition [31]. The key idea is a common set of competition goals in which researchers benefit by comparing algorithms, hardware and performance.

The RoboCup competition has levels which only compete *standard platforms* to remove the problem of building physical hardware. A standard research platform is essential to compare algorithms and verify performance of a robotics system. Examples of successful robot platform standardization come from the AIBO dog [24] and the NAO humanoid [32] shown in Figure 2.4(a), 2.4(c). Both platforms through RoboCup developed a standard league competition which rapidly progressed machine learning and the multi-agent behaviors [26].

RoboCup also features a small wheeled multi-robot competition called the Small-size League. These small-size platforms typically lack distributed sensing and communication. One of the most well known small-size robot designs in RoboCup is the 2006 design from Carnegie Mellon as shown in Figure 2.4(b) [25]. However, these small-size platforms are designed for robot soccer and not fully distributed multi-robot research.

2.3 Multi-Robot Coordinate Systems

This work is interested in the defining a new angular based coordinate system. Existing coordinate systems used in robotics have varied between global, local, bearing-only and range-only as shown in Figures 2.5(a)- 2.5(d).



(a) Global. (b) Local. (c) Bearing-Only. (d) Range-Only.

Figure 2.5: **a:** Global coordinates. **b:** Local Coordinates in frame of blue robot.
c: Bearing-only Coordinates in frame of blue robot.
d: Range-only Coordinates in frame of blue robot.

Bearing-only sensors on the r-one sets the stage for an angular coordinate system. Much of the previous work on computation of coordinates for multi-robot systems focuses on computing global coordinates for each robot in the group using beacon or anchor robots with fixed GPS coordinates [33, 34]. There are also distributed approaches, which do not require globally accessible beacon robots, but instead use multi-hop communication to spread the beacon positions throughout the network [35]. These approaches breakdown in GPS-denied environments and will not scale for large swarms of mobile robots [36].

Related Network Geometry Approaches.

The algorithm presented is focused on bearing information and formulating scale-free coordinates between robots. Obtaining accurate local network geometry regardless of the scale of the system is a critical feature in a multi-robot configuration control. The literature presents multiple approaches to network geometry such as pose in a shared external reference frame [37], pose in a local reference frame [19], range-only [38, 39] bearing-only [40, 41, 42], and even only the sorted order of bearing [43].

2.4 Centroid Experiments

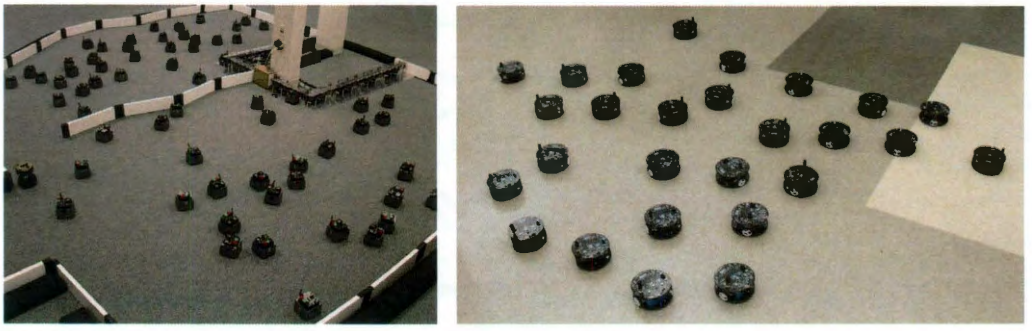
To verify the performance of scale-free coordinates, a series of experiments validate *centroid behaviors* using the r-one. A centroid behavior consists of a robot moving to the centroid of an arbitrary shape of neighboring robots. This work compares centroid experiments with related *limited coordinate system experiments* in which a robot does not have full local Pose information of the neighboring robots. Limiting the coordinate system information is related to *minimal sensing*, in which a system only has the minimum sensing information to accomplish a task.

In the early days of robotics, low-resolution sensors was a primary option for detecting obstacles. Robots accomplished basic tasks with limited robot capabilities such as Shakey [44] and the Stanford Cart [45]. In more recent times, high-resolution sensors and computational resources are becoming a standard in robotics research. However, multi-robot systems limited in size and cost are still interested in the idea of scaling back to minimal sensor capabilities to accomplish a goal. Limiting the sensing capabilities also limits the coordinate system information and potential tasks a robot can accomplish. Erickson et al. presents a blind robot with only a bump sensor and timing clock to solve robot localization in a complex environment [46]. This work breaks down the coordinate system of the world to only points along the boundary of the environment and ignores all other points inside the boundaries. Limiting the coordinate system model of the world to only boundaries enables their robot localization algorithm to be successful. Their approach only relies on the robot's initial orientation, a clock for tracking movements and a bump sensor for obstacle detection. With a known map of the environment, the robot starts in any random place and begins to move into walls around the environment. The robot begins to localize and continuously improves its localization estimate as it moves through the world. An additional discussion of limited coordinated coordinate experiments is provided in Chapter 6.

Chapter 3

The r-one robot

This work introduces a new multi-robot system, the *r-one* shown in Figure 3.1(b). The design of r-one was focused on building a fully distributed low-cost robot. The design requirements also emphasized a small size, odometry, LED display, push buttons, line-of-sight and omni-directional communication. Many of these requirements were lessons learned from the SwarmBot design from McLurkin [19] as shown in Figure 3.1(a). For example, the SwarmBot featured four circuit boards with a microcontroller and FPGA. The goal of the new robot design focused a reduction to two circuit boards and one main microcontroller without an FPGA.



(a) SwarmBots. (b) r-ones.

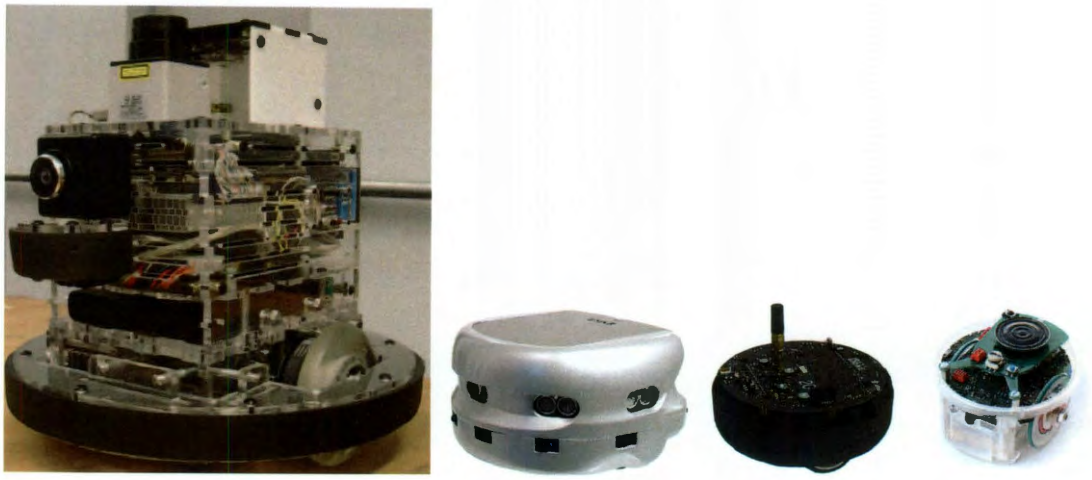
Figure 3.1: **a:** SwarmBots buit by iRobot and McLurkin in 2003 [19].
b: R-one robots built at Rice University with lessons learned from the SwarmBots.

The r-one design team searched for available COTS hardware for multi-robot systems. The 3pi from Pololu laid the actuation roadmap for our single circuit board design with integrated low-cost motors. Unfortunately, there was not an available platform with sen-

sors and actuation that met our multi-robot system requirements. Therefore, the team invested in designing a custom electrical and mechanical design. The main focus of the search was selecting microcontrollers, sensors, motors and connectors. During extensive evaluation of electronic parts, the r-one design went through eleven design prototypes. During the process of design, related approaches to multi-robot systems were compared.

3.1 Related Multi-robot Systems

Four multi-robot platforms are considered in this work: the Khepera III, the E-puck, the Scarab and the r-one as shown in Figures 3.2(a)- 3.2(d). The Khepera III was built by Mondada at K-Team [47] and the subsequent E-Puck was developed when Mondada moved back to EPFL [48]. The Scarab is built by Michael and Kumar at University of Pennsylvania [49]. Most recently in 2010, the r-one was introduced by McLurkin and Lynch at Rice University [5]. All of these platforms are vastly different in cost, capability and hardware. Despite the developments in different platforms, a standard platform for multi-robot research is still in the beginning stages [49]. To date, a practical low-cost multi-robot system has not gained wide adoption in the robotics community [5].



(a) Scarab, 25cm (b) Khepera III, 13cm (c) R-one,11cm (d) E-puck, 7cm

Figure 3.2: **a:** The Scarab robot designed by University of Pennsylvania [49]. **b:** The Khepera III robot produced by K-Team [47]. **c:** The r-one robot presented by Rice University. **d:** The E-puck robot designed by EPFL [48] is a fully distributed multi-robot. T

3.1.1 Platform Comparisons

r-one. The r-one multi-robot system is covered in the subsequent chapters. However, the r-one has a full suite of sensors including encoders, gyro, accelerometer and infrared system for inter-robot communications. The infrared system also can detect obstacles for localization of the robot. The $11cm$ r-one is a low-cost solution for educational outreach and research at $245 cost per robot. In addition to capable hardware, the robot supports either Python or C programming. The embedded python environment makes this robot especially attractive to introductory programming courses.

Khepera III. The Khepera III is a commonly used multi-robot system in the research community. The $13cm$ robot is manufactured by K-Team Corporation with a sensor suite that includes encoders, nine infrared range detectors, five ultrasonic range sensors and two cliff detectors. The robot features a DsPIC 30F5011 at 60MHz with ability to expand to a KoreBot gumstix processor [47]. In addition to the internal sensors, Khepera is designed

to support modular extensions such as grippers and additional sensors. The cost of this multi-robot system is prohibitive for large populations at approximately $\$4,270$ per robot.

E-puck. The E-puck multi-robot system was developed by EPFL [48]. E-puck designer F. Mondada started with the Khepera group and moved onto make a simpler education outreach tool for universities. The $7cm$ e-puck is equipped with encoders, VGA camera, 3 omni-directional microphones, 3-axis accelerometer and 8 IR proximity and ambient light sensors. An optional infrared range-bearing turret gives the e-puck distributed sensing and communication. This turret is not built into the robot like the r-one infrared system but is an additional module which puts the cost of one robot at $1388. The main limitation of the e-puck compared to the r-one is the triple increase in cost. When developing large populations of robots, a low-cost platform is essential to scale the population to dozens of robots.

Scarab. The Scarab is built by Michael and Kumar at University of Pennsylvania [49]. This robot is significantly larger at $25cm$ diameter and a $8kg$ weight. This robot is also in a different computational class with a PC computer and a Point Grey Firefly IEEE 1394 camera and Hokuyo URG laser. This robot has the sensor and computation payload for simultaneous localization and mapping, SLAM. However, this robot is not feasible or practical for large populations due to the $1500 base platform cost, along with an additional $400 camera and $1100 laser. The Scarab robot presents a shift away from low-cost sensing on multi-robot systems. Instead the Scarab's PC computer and advanced laser and camera sensors draws from the successful outdoor robot research vehicles. The Scarab design shifts away from minimal multi-robots like the r-one and instead adds a complex laser sensor system.

3.1.2 Cost vs. Functionality

This work is interested in a minimalistic sensing approach to solving common multi-robot behaviors. Reducing our cost of the robot to a minimum to achieve suitable multi-robot

performance is ideal. An effect of reducing robot cost is the reduction of robot capabilities. For example, the r-one removed the cost of range-bearing sensors and included bearing-only sensors. Range-bearing sensing is important for flocking and precise formation control. However, bearing-only sensors are less complex and still capable of some types of multi-robot configuration control and localization.

Our current r-one robot costs $245 in quantities of 1000 or more. The component costs total $163 and PCB fabrication is $15. The assembly of the circuit boards is $67 as shown in Figure 3.3.

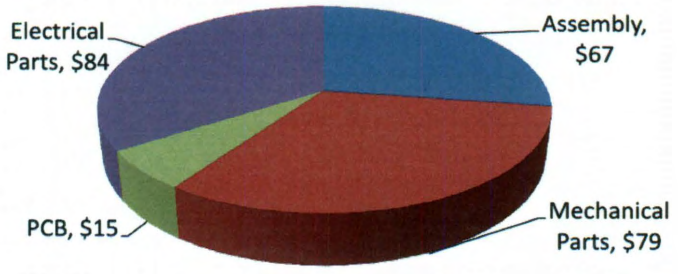


Figure 3.3: Breakdown of r-one assembly, printed circuit board (PCB), electrical and mechanical part costs. These part and assembly costs are in estimates of 1000 or more robots.

The e-puck and the r-one share similarities in sensor suite and physical size. Cost is the major difference between the two platforms. The e-puck features an infrared turret for range-bearing sensing compared to the r-one's bearing-only sensing. However the e-puck with range-bearing turret is still not a low-cost robot. The r-one's main contribution to the multi-robotics community is an integrated, low-cost platform with inter-robot communications, a sensor for network geometry, a system for ground truth position and a flexible embedded python development environment.

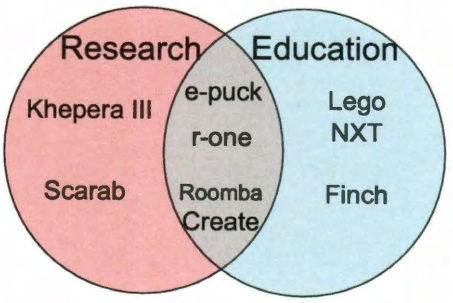


Figure 3.4: The educational and research space of small mobile robotics platforms. The e-puck and r-one are designed for research and educational multi-robot activities. The Roomba Create is a low-cost robot platform being used by many researchers and educators.

3.1.3 Research and Education Robots

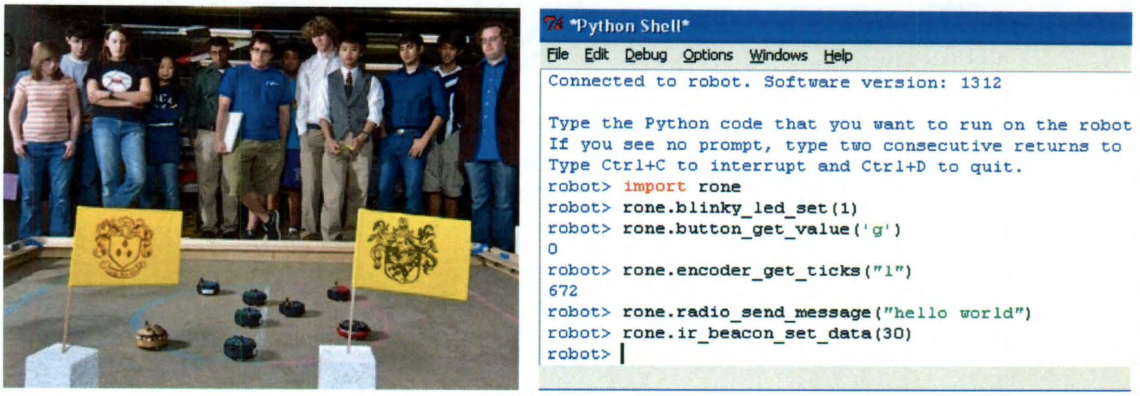
The r-one multi-robot is a platform capable for research and education. Therefore, the r-one will be compared against fully distributed multi-robot systems and educational robots. Educational robots are also a useful cost comparison to see the minimum capabilities of available robot systems. However, the educational platforms such as Finch and LEGO NXT are not considered multi-robot systems. Figure 3.4 illustrates the intersection between research and education of the r-one, Roomba create and e-puck robots.

Table 3.1 compares the differences between these platforms. The Pololu 3pi and Finch are inexpensive, but lack basic sensors, such as wheel encoders, and do not have the communication systems required for multi-robot coordination. LEGO Mindstorms is the leader in educational robotics, but lacks a sensor for detecting local network geometry, *i.e.*, the positions of neighboring robots. The iRobot Create is a popular platform for medium-sized robots, but the size, cost, and limited sensor suite require many add-on components for multi-robot work. The robomote and the costbots are not considered fully distributed multi-robot systems because they lack the sensors needed to determine local network geometry.

Robot	Source	Wheel Encoders	Radio	Neighbor Position	Robot ID Beacon	Visible Light sensor	IR Range	Ultrasonic Range	Accelerometer	Gyro	Bump Sensor	Cliff Detector	Temperature Sensor	Camera	Microphone	Remote Programming	Gang/Self Charging	Other features	Retail Price ($)	Parts Cost ($)
Khepera III	K-Team	0	0				0	0			0								2000	-
Create	iRobot	0								0	0								220	-
Mindstorms	LEGO	0	0			0		0		0									249	-
Finch	Finch					0	0		0			0							99	-
e-puck	EPFL	0	0	0			0		0				0	0	0				1388	-
3pi	Pololu										0							IR line(x5)	99	-
CostBots	Berkeley		0						0									NEST sensor	-	200
Scarab	uPenn	0	0										0					laser range	-	3000
robomote	USC	0	0				0			0								compass	-	150
r-one	Rice	0	0	0	0	0	0		0	0	0	01	0			01	01		-	245

1 = Currently in development

Table 3.1: A comparison of available low-cost robots suitable for multi-robot research. The research robots are provided in terms of part costs. The commercialized robots are based on prices available to consumers.



(a) Class with r-one.

(b) Python prompt on r-one.

Figure 3.5: **a:** The Introduction to Engineering class from Rice University using the r-one.
b: The embedded python interpreter on the r-one.

3.1.4 r-one in Education

The r-one has been used successfully in a introduction to engineering class at Rice University as shown in Figure 3.5(a). The class was structured around teaching python on the r-one to allow students to adjust a motor or measure sensors with a high-level API. Embedded python coupled with this r-one API allows the students to accomplish complex tasks with a relatively small amount of code. This increases students' ability to focus on solving the problem rather than on programming complexities. The students can also verify commands and syntax on the python prompt on the robot as shown in Figure 3.5(b). In the class at Rice, students were able to implement a velocity controller and follow-the-leader behavior without prior programming knowledge. The class has gone through two semesters with promising results for students and the future of the r-one in education.

(a) r-one. (b) The r-one encoder. (c) Exploded View.

Figure 3.6: **a:** The r-one robot was designed by MRSL group at Rice University for multi-robot research, teaching and outreach. **b:** The motors and encoders mount directly to the circuit board with custom-built encoders to reduce cost. **c:** Exploded CAD view of the robot shows the simplicity of the parts and assembly process. The assembly consists attaching two circuit boards and a plastic shell with screws.

3.2 Robot Specifications

The r-one has a full sensor suite including a gyro, accelerometer, encoders, light and bump sensors. The r-one is 11 *cm* in diameter and weighs approximately 305 *grams*. It's on-board computation features a 32-bit Texas Instruments Stellaris LM3S8962 ARM Cortex-M3 microcontroller running at 50 mhz with 64 KB of SRAM and 256 KB flash storage. The r-one also includes 15 display LEDs, audio MIDI processor, SD card, infrared (IR) and radio communication. The local IR communication system is the primary means of inter-robot communication and localization. The 2.4 GHz radio on the robot can be used for inter-robot communication, but is designed for centralized command and control.

The exploded robot CAD diagram is shown in Figure 3.6(c). The robot is composed of two circuit boards bound together with a circular shell. The shell also serves as a protective shield to channel IR sensor measurements and integrated springs to facilitate a bump skirt for the 8 bump sensors distributed around the perimeter of the robot.

Top Circuit Board.

The top circuit board contains the user interface and the microcontroller. The user interface includes 3 push buttons and 3 arrays of five LEDs each in red, green, and blue. Each of the 15 total LED elements has individual brightness adjustment controlled by a Texas Instruments LED driver (TLC59116F) over I^2C. I^2C is a bus protocol designed for communication between two electronic devices. For audio feedback, the robot uses the audio MIDI driver made by VLSI(VS1053) with a custom analog circuitry for driving an on-board 0.5 $Watt$ 40mm speaker. The 2.4Ghz radio is controlled by a Nordic controller NRF24L01P using SPI commands. SPI is another type of bus protocol to communicate with multiple electronic devices. The radio is designed for 2$Mpbs$ bandwidth and has been verified to work reliably at a 15$meter$ range. The top circuit board also includes 8 IR transmitters and 8 IR receivers made by Sharp (GP1UX311QS). The geometry and characterization of the IR is discussed in the subsequent section.

Bottom Circuit Board.

The bottom circuit board includes the motors, quadrature encoders, 3-axis gyro, 3-axis accelerometer and 8 bump sensors. The bottom board also has a **MSP430**F2132 16-bit microcontroller with 8K Flash and 512 Bytes of RAM. The MSP430 provides additional input-output to the main Stellaris microcontroller on the top circuit board. This MSP430 microcontroller interfaces with the 3-axis gyro (L3G4200D), 3-axis accelerometer (MMA8452Q), the bump sensors and the LED Driver on the top circuit. Figure 3.7 illustrates the system communication between the sensors in green, interfaces in red and microcontrollers in blue on each respective circuit board. Figure A.3 shows complete block diagram of the system with power and information data connections.

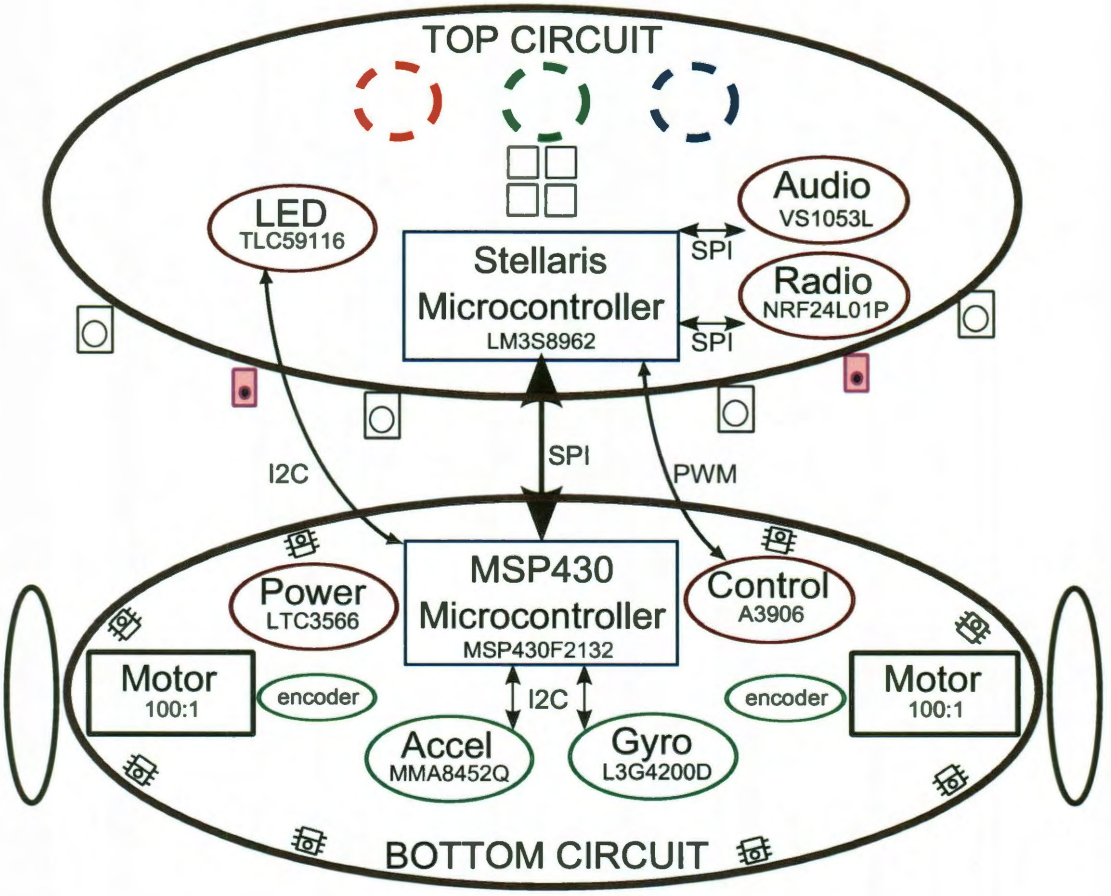
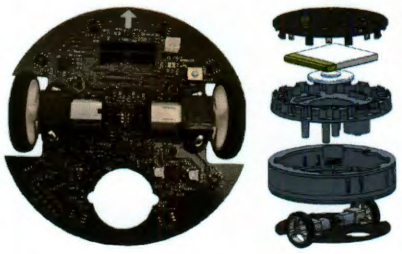


Figure 3.7: A system block diagram of devices on the r-one robot. The top and bottom circuit boards feature microcontrollers in blue, sensors in green, interface devices in red.



(a) Bottom Circuit. (b) Mounting Hardware.

Figure 3.8: **a:** The bottom circuit board with eight bump sensors and integrated motor encoder units. **b:** Mounting hardware has three sections. The shield, support and skirt.

Bump Skirt.

The robot features a plastic bump skirt with eight bump sensors to detect collisions in any direction. The bump sensors (SFH9240-Z) use an optical reflection technique to bounce light off the bump skirt when an obstacle moves the skirt. The bump sensors are placed around the robot to provide awareness of collisions from any direction as illustrated in Figure 3.7. The skirt is part of the three plastic parts between the two circuit boards as shown in Figure 3.8(b). The first part is the IR shield, the shield provides the geometry characteristics of IR communication between robots and obstacles. The shield also provides the support standoffs and mounting holes for the circuits. The other two plastic parts include the skirt exterior and the spring section which interlock together to form the cylindrical robot shape. The plastic spring section in the skirt have enough stiffness to avoid triggering the bump skirt while the robot is moving. However, being too stiff will prevent the robot from detecting collisions with non-rigid obstacles. Thus, a careful balance in the spring stiffness was the pivotal part of the bump skirt design.

Motor-Encoder Performance.

The motors and encoders mount directly to the bottom circuit board, shown in Figure 3.6(b). These low-cost quadrature encoders are a new design, and use an optical

Operation	Current (mA)	Duration (hours)
Airplane mode	0.060	30000
Standby (no LEDs, Motors)	250	7
Moving and LEDs	610	2.5
Stall Torque	750	1.5

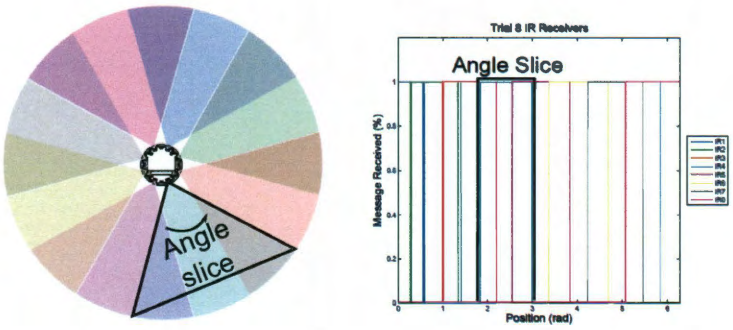
Table 3.2: Current consumption at different modes of operation for the r-one robot. These current numbers and time duration are approximate depending on the 3.7 2000mAh Lithium Polymer battery performance.

interruption sensor to detect gaps in a custom encoder wheel attached to the rear motor shaft. The encoder discs are made from plastic on a laser cutter, and manufacturing tolerances limit the design to four slots, producing a 0.625 mm/tick linear resolution at the wheel. The two motors are controlled by an Allegro A3906 motor controller over pulse-width modulation, PWM. The 32 mm wheels coupled with a 100:1 gearbox give the robots a maximum speed of 300 mm/sec, while internal friction in the gear-train limits the minimum controllable speed to 15 mm/sec.

Power Performance The robot uses a DC-DC Buck-Boost power regulator from Linear Technology (LTC3566). The 3.7V 2000 mAh lithium-polymer battery is regulated to 3.3V as the primary voltage on the robot. The Total system power is 250 mA without activating motors or LEDs. Under stall torque with an active LED group, the robot can peak at 750 mA. In normal movement operations with LEDs active, the robot consumes 610 mA as shown in Table 3.2. The robot can perform a motion experiment for an average of 3 hours of run-time. When powered off, the robot enters airplane mode and draws $60\mu Amps$. The robot only stops consuming power when the battery is physically disconnected. The power regulator also features circuitry to charge the battery from a USB port and the self-docking charging prongs on the top of the circuit board.

3.2.1 Infrared Communication

Each robot has eight IR transmitters and eight receivers. The transmitters broadcast in bursts and emit a radially uniform energy pattern. The robot's IR receivers are radially spaced to produce 16 distinct detection regions as shown in Figure 3.9(a). By monitoring the overlapping regions, the bearing of neighbors can be estimated to within approximately 22.5°.



(a) IR receiver overlap. (b) Measured IR overlap.

Figure 3.9: **a:** A top view of the r-one's IR receiver detection regions. Each receiver detects a 68°angle slice which overlap to form 16 distinct sectors. This allows a robot to determine a neighbor bearing within 22.5°.

b: Experimental verification of the overlap of the receiver regions. The plot illustrates the angle each receiver detects an incoming message. The average angle width of detection is 68°. The corresponding arc from the top view in Figure(b) is highlighted in black.

The error model of the IR communication is a function of the collisions with the physical IR shield and the transmitter power. The shell is designed to limit the detection of a single IR receiver to a 68°arc. Figure 3.9(a) demonstrates the measured reception arc from each receiver with color corresponding to the sectors from Figure 3.9(a). The arc varies from 63°to 74°with a mean of 68°over 10 trials. This measured mean matches the initial design of the shield and provides a physical model of the IR receivers' geometric

Start	Data	Orientation	CRC
1-bit	16-40 bits	8-bit	16-bit

Table 3.3: A message layout from start to finish. The data bytes can be varied from 3-bytes to 5-bytes.

tolerances. The IR receivers are Sharp IR remote control devices with 38khz modulation and a maximum bit rate of 1250bps. The IR communication is intentionally range-limited to approximately 1.5 meters to reduce the interference between neighboring robots. A time division multiple access (TDMA) protocol is used to transmit messages to every fixed communication interval with a random start delay similar to the ALOHA protocol [53].

The IR message transmitted is an encoded message with a start and checksum bits as shown in Table 3.3. A 16-bit cyclic redundancy check (CRC) is utilized in the encoding scheme to detect errors in the communication process. When communicating with a network of robots, a *network round time* determines the period in which each robot can communicate to the network. The equation for calculating network round time is shown in the equation below

$$Time_{round} = 2(1 + N)^2(time_{bytes})$$

The network round time $Time_{round}$ is based on the number of neighbors, N and the transmit time per bytes $time_{bytes}$. The approximate message sizes and transmit time is shown in Table 3.4. The scale-free experiments in this work use a 3-byte message and a maximum of five neighbors, which totals to a 2.5 second network round time. The network round communication limits the physical speed of the robots in the network. A potential upgrade in the recievers bandwidth would alleviate the robot speed constraints placed on this network. However, robots moving at fast speeds is not a primary goal of this research.

Message Size (bytes)	Message Size (bits)	Transmit Time (ms)
3	41	32.8
4	49	39.2
5	57	45.6

Table 3.4: A mapping of messages size in bytes to the actual transmit time for a robot. The transmit time scaled for an average of 6 robots in the network is approximately the network round time.

3.2.2 Network Round Communication

Each robot in the network will broadcast a message to all its neighbors in a round. The network progresses as a synchronous distributed system from each robot's point of view as stated in McLurkin [19]. This concept works as long as all robots share the same δt update period between all robots as shown in Figure 3.10. When a new round begins, each robot clears all prior neighbor information and updates the network information again. The chances of collisions are high when the number of robot messages approaches the limit of δt. Even when the δt update period is much larger than the total messages on the network, collisions can still occur. Thus, a randomized start delay, d_B, d_C is added to further reduce the chance of robots colliding on messages as shown in Figure 3.11.

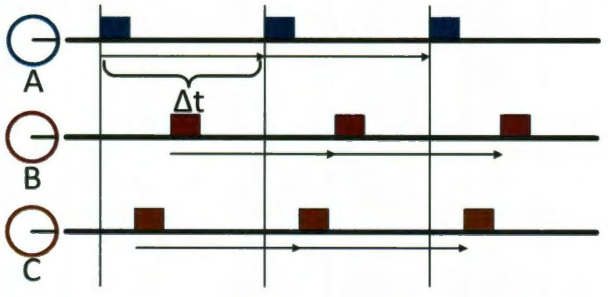


Figure 3.10: Neighbor Period with complete synchronization between neighbors.

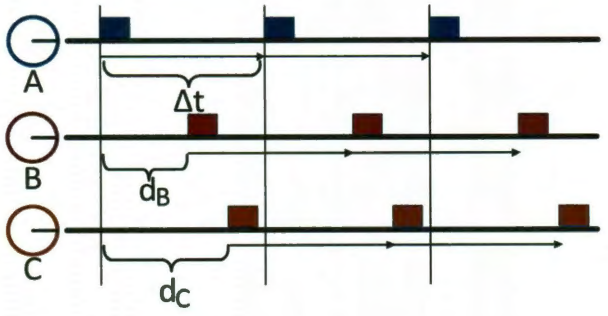


Figure 3.11: Neighbor Period with randomized starting between neighbors. The randomized delays for B and C are depicted by d_B and d_C.

3.2.3 Sensing Orientation of Neighbors

Sensing a neighbor's *orientation* in local coordinates requires knowledge of *bearing-to-neighbor* and *neighbor-bearing-to-sensor*. In other words, orientation is the angle from the neighboring robot back to sensing robot. Figure 3.12 depicts orientation, bearing and range of a red neighbor robot and a blue sensing robot.

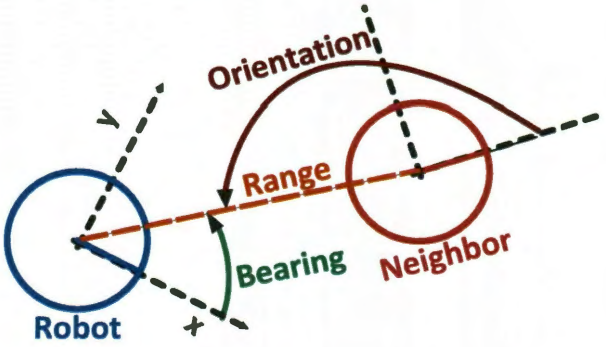


Figure 3.12: Bearing , range and orientation of a red neighbor robot from a blue sensing robot.

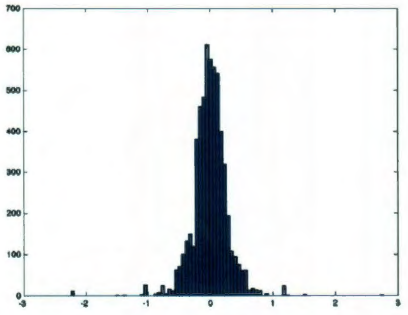
The first version of the r-one employed a *reciprocal orientation* method to measure neighbor orientations. This methods requires a significant amount of communications to obtain orientations for each neighbor. Each r-one broadcasts an *announcement message*, which includes its respective neighbor ids and bearings. Then for each neighbor, the robot will broadcast a corresponding message. When a robot receives an announcement message, it can calculate the orientation of the announcement message if its ID is included in the

message. The communications cost is high because each r-one transmits an announcement message plus an additional message for each neighbor. This method has a computational complexity of $O(1 + N)$ where N is maximum number of neighbors.

The second version of orientation measurement uses an *encoded orientation bit* in the transmission hardware. This method sends a unique communication bit from each transmitter on the robot. Therefore, the announcement message has the orientation information encoded into one-bit in the message. This technique at first was not possible on the r-one due to hardware limitations that did not allow unique messages from each transmitter. Before, the robot had to send the exact same message on each transmitter. Now, the encoded orientation bit is the primary method utilized for orientation sensing on the r-one robots. This reduces the communications complexity significantly by reducing the total number messages to exchange between robots.

Validation of Orientation Sensing.

Orientation sensing was validated by measuring the local estimated neighbor orientation versus the global coordinate headings. Global headings were obtained through collecting data with a tracking described in the subsequent section. The histogram of bearing accuracy for 10 trials between 1-4 neighbors is shown in Figure 3.13(a).



(a) Bearing Errors.

Figure 3.13: **a:** The bearing error for 10 trials of 1,2,3,4 neighbors resulting 50 total experiments.

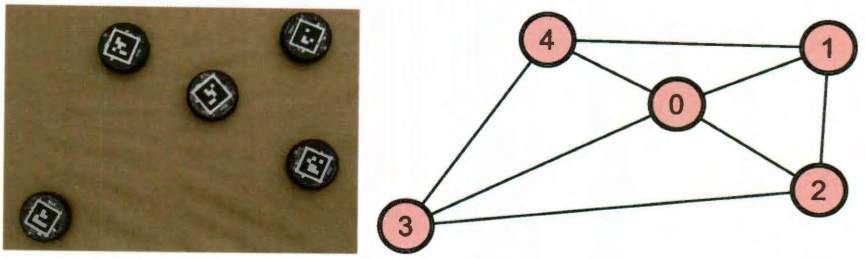
3.3 Data Collection

Data collection on multi-robot systems is the process of collecting global positing information on the robots. Global can be defined as the global space in the room or as GPS coordinates. For the purpose of this work, global is referred to as the *ground-truth* positions of the robots. There are many means of determining a robot's ground-truth position in the literature: GPS [54], a beacon tracking such as Vicon [55], radio-acoustic ranging [56, 39], or camera-based tracking [57, 58, 59, 60]. GPS is unavailable indoors and a Vicon system is too expensive for a low-cost multi-robot platform. Radio acoustic systems are a viable indoor tracking system, but increase the complexity of each robot.

Camera-based tracking systems are currently the most common low-cost method for determining a robot's location in an indoor environment. These systems must have the ability to uniquely identify individual robots in the camera image.

Camera tracking robots with *fiducial* markers is a popular technique and used most prominently in RoboCup [57]. In a uniform environment, robots can be tracked by color alone as with SwisTrack [58]. Bar code tags such as AprilTags [59] also provide unique IDs without initialization, and 6-DOF pose estimation. However, bar codes require a vast collection of image processing tools. A less complex tracking method involves tracking *active IR LEDs* mounted on each robot [55, 60]. The active IR LEDs transmit a pattern unique to each robot. One active IR LED per robot and one camera allows 2-DOF position to be measured directly. Multiple cameras or beacons can be used for full 6-DOF pose estimation [61].

The r-one robots are internally designed for IR LED and APRIL bar-code tag tracking. The performance of active IR LED tracking has also been characterized to work at 7.6 *meter* distance between robots and camera. The IR tracking camera has a significantly larger viewing area than the 1.2 *meters* of distance between the APRIL camera and the robots. However, the APRIL tags system provides heading (x, y, θ) of each robot while the IR LED tracking only provides positions (x, y).



(a) 5 robots in configuration. (b) Ground Truth estimate by simulator.

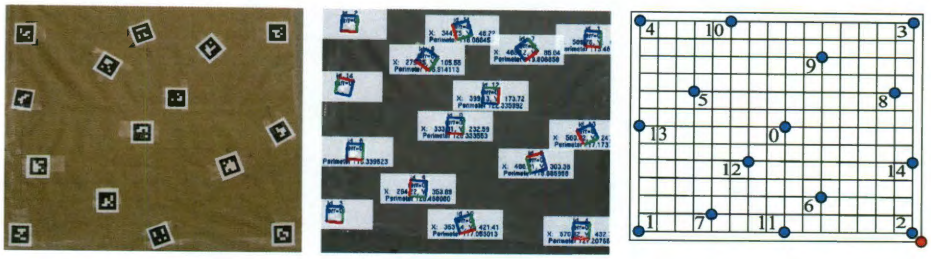
Figure 3.14: **a:** An image from the overhead camera from our data collection system. The five robots in the picture are outfitted with APRIL tags for detection of ground-truth 2D position and angular heading. **b:** Visualization of ground truth camera data. Edges are denoted by IR communication links between robots.

For the experiments in this work, we collect ground-truth pose information using the APRIL tags software system [59]. Figure 3.14(a) displays the camera's perspective of the robots in a typical experiment. To extend our field of view, two cameras were used in parallel to track the robot workspace.

3.3.1 Data Collection Calibration

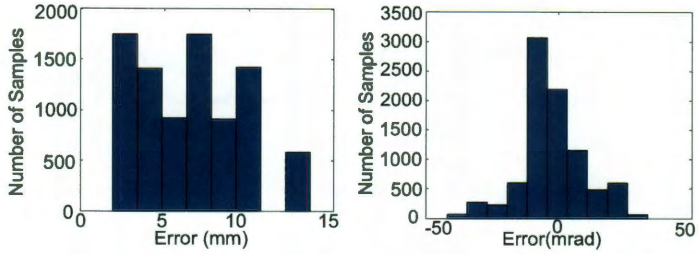
The APRIL tag system relies on each robot having a unique identification tag which provides a tracking element for the image processing algorithm. The APRIL tags system outputs each robot's Pose (x, y, θ) in 2D euclidean space. Figure 3.14(a) displays the camera's perspective of the robots in a typical experiment. For calibration, we created a workspace with 15 robot tags and took accurate physical measurements for position and orientation.

The distance error histogram for the calibration test is displayed in Figure 3.16(a) and has a mean error of $6.56mm$. We complete 2D position calibration of the robot with an orientation error histogram with a mean error of $9.6mrad$ as plotted in Figure 3.16(b). These trial calibration runs were performed 583 times to provide a reasonable error estimate for each of the 15 robots spread across the workspace.



(a) Raw Camera Data. (b) April Tags Overlay. (c) Calibration of 15 robots.

Figure 3.15: **a:** The data collection's overhead camera image. **b:** An overlay-ed 2D position for each unique tag. **c:** The plot displays the estimated positions of 15 robots in the workspace for camera calibration. The grid is based on 2 inch cells.



(a) Distance Error. (b) Heading Error.

Figure 3.16: **a:** The distance error from APRIL tags for each of the 15 robots over 583 samples with a mean of $6.56mm$. **b:** The corresponding heading error for each robot with a mean of $9.6mrad$.

3.3.2 Collection Data Flow

Figure 3.17 shows a diagram of the complete data collection system for the r-one. The system will have four main components: the robots running the experiment, a single host robot, a localization camera connected to APRIL tags server, and a client computer with data logging software. Ground truth positions of the robots are measured the vision system to track the $\{x, y, \theta\}$ positions of all of the robots simultaneously. The server will collect and display all camera estimates of individual robot positions based on unique IDs.

The host robot receives wireless messages from the robots running the experiment and

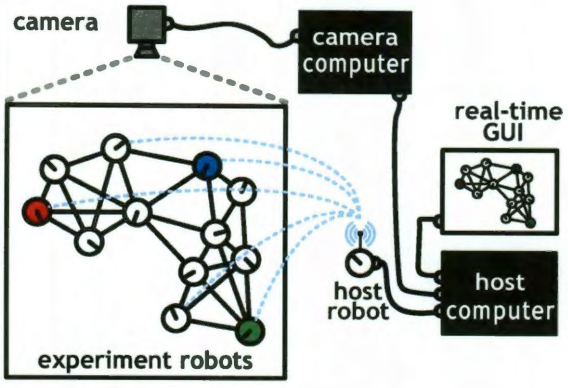


Figure 3.17: Diagram of the r-one data collection system. The ceiling-mounted camera identifies each robot and tracks the $\{ x, y, \theta \}$ positions of each robot simultaneously, reporting the results to the computer at 5 Hz. A larger version of this figure is available in the Appendix section.

prints the IR neighbor communication information in CSV format over the USB terminal. The data logging client computer connects to the APRIL tag server and the host robot merges together robot communication and ground-truth positioning information. The Java client software enables data logging of experiments and real-time display of the robot geometry and communication links. Figure 3.17 demonstrates a real-time screen-shot of an experiment with a ground-truth camera. The IR communication links between the robots are shown as black edges.

In future experiments, the visible robot workspace is scalable with more computers and cameras. Each computer is capable of processing two cameras and streaming out position information over the network. The host computer has the ability to collect network information from multiple computers and scale the workspace. Also, if only 2D (x, y) position is required, the single IR tracking camera has a workspace that is approximately six time larger than a single APRIL tag camera.

Chapter 4

System Model and Definitions

In this chapter, I present a series of definitions and models to lay the foundation for the mathematical notation used in the simulations and algorithm.

4.1 Communication Network

The inter-robot IR communication network is modeled as an undirected graph, $G = (V, E)$ where $E(G)$ and $V(G)$ denote its set of vertices and edges respectively. The neighbors of each vertex u are denoted by $N(u) = \{v \mid \{u, v\} \in E\}$. Figure 4.1 depicts the connected edges of communication between robots. The labeled e represent a edge subset of Robot 0's neighbors, $N(0)$. The *realization* of graph G is defined as a function $p : V(G) \to \mathbb{R}^2$ that maps each vertex of G to a point in the Euclidean plane.

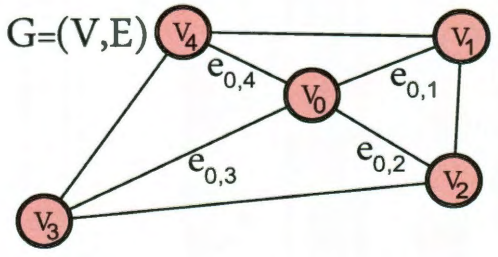


Figure 4.1: Model of undirected graph neighbor edges of vertex 0 highlighted as $N(0)$.

To represent the k-neighborhood of robot u we introduce N_u^k to denote the closed k-neighbors of u: the set of vertices reachable by paths starting at u and of length at most k. In Figure 4.1 the N_0^k is marked for the central robot 0. We can remove all edges and vertices not connected to robot 0 to form a sub-graph G_0^k as the k-neighborhood of robot

0.

4.2 Connecting Local Coordinates to Scale-free

There are three coordinate systems denoted by positions $p = (x, y, \theta)$ of particular interest in this work. The first coordinate system is ground truth position of robot u in a global coordinate system denoted as $p_0(u)$. The second coordinate system is the local coordinates of each robot which is estimated positions, $p = (x, y, \theta)$ from a local robot perspective. With ideal range-bearing information, local coordinates would be trivial to estimate without the use of scale-free information. However, given our bearing limitations in resolution, we have introduced the local scale-free as the third coordinate system to represent the position p of the robot.

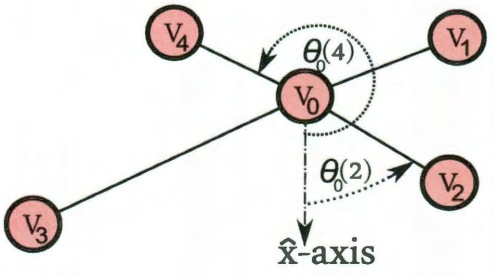
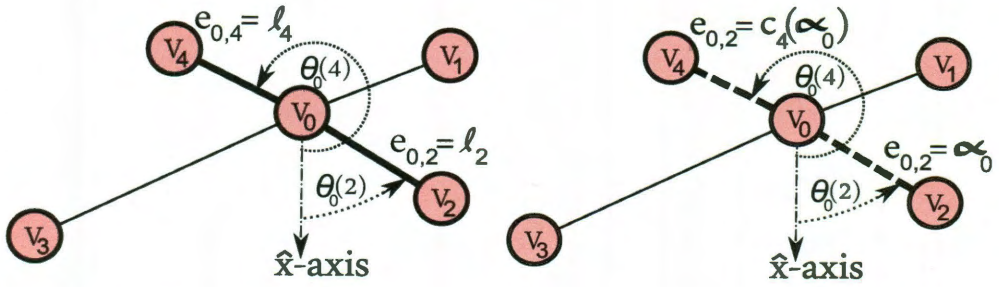


Figure 4.2: An illustration of the angles of robot 2 and 4 from robot 0's perspective. The other angles are omitted for ease of illustration.

In both local coordinates and local scale-free coordinates, a robot sits at the origin of its coordinate system. The robot's $\hat{x}$-axis is aligned with its current heading. Each robot can measure the exact angle to its neighbors with respect to its $\hat{x}$-axis. More formally, a robot u can query a function $\theta_u : N(u) \rightarrow [0, 2\pi]$ that maps every neighbor $w \in N(u)$ to the counter-clockwise angle to w from the $\hat{x}$-axis of u's coordinate system. Thus $\theta_u(w)$ represents the angle measurement towards neighbor w returned by robot u's sensors as shown for robot 0 in Figure 4.2.



(a) Local Coordinates.

(b) Scale-free Coordinates.

Figure 4.3: **a:** Local coordinate system illustrating the length edges between neighboring robots $w = 2, 4$ of robot 0. **b:** Local scale-free coordinate system illustrating the length edges as a function of α_0 and coefficients c_w between neighboring robots $w = 2, 4$ of robot 0. In this case $c_2 = 1$ because it is the first neighbor encountered from counter-clockwise perspective of $\hat{x}$-axis.

The difference between these two coordinate systems is in how robots perceive distance. When using local coordinates, robot u queries a function $\ell_u : N(u) \to \mathbb{R}^+$ which maps every node $w \in N(u)$ to the distance from u to w, that is $\ell_u(w) = \|p_0(u) - p_0(w)\|$. On the other hand, in local scale-free coordinates the function ℓ_u maps every node $w \in N(u)$ to the *scale-free distance* from u to w. Formally, there exists an unknown constant $\alpha_u > 0$ such that for every $w \in N(u)$ we have $\ell_u(w) = \alpha_u \|p_0(u) - p_0(w)\|$. Therefore, local coordinates can be regarded as a special case of local scale-free coordinates where α_u is unitary and known to every robot u. For illustration of the concept, Figure 4.3(a) depicts the difference between local coordinates and scale-free coordinates. Only two edges are considered to demonstrate that scale-free assigns the first robot to a α_u scaling factor and the subsequent neighbor edge lengths follow a coefficient of alpha $c_w(\alpha_u)$.

Chapter 5

Computing Scale-free Coordinates

In this chapter, I outline a procedure to compute scale-free coordinates and compare against related coordinate systems. The full procedure is derived in detail in Cornejo [62], but the summary here provides a foundation for this paper. The full procedure has been verified in simulation and theory. However, the techniques employed in the full procedure are not feasible for computation on microcontroller-based robots. Thus, I also present a less complex algorithm considering only 2-hop communication and triangle cycles between neighboring robots.

5.1 Related Coordinate Systems

This section provides a more detailed comparison between scale-free coordinates and related coordinate system work. Scale-free coordinates enhances bearing-only sensor model to provide a more dominant coordinate system. Related approaches use only range, bearing and noisy combinations of both.

Range-only Approach.

The "robust quads" work of Moore et. al. [38] is the closest in problem formulation to this work. Using only distance information, they use *robust quadrilaterals* in the local communication network to produce local network geometry for each robot. *Robust quadrilaterals* utilize a quadrilateral between four communication nodes. This work is in a similar spirit, since bearing information is used in the network to produce local network geometry. In the error-free case, this paper presents localization success rates which are comparable to the Moore results.

Bearing-Range Approach.

There is also related work on the problem of localization with both bearing and distance information. For example, Basu et al. [63] studies the problem of localization assuming nodes have noisy distance and angle measurements/constraints. The work of Dogancay [64] studies localization with a static observer and a moving target to which the observer can measure a bearing, which is analogous to solving a triangulation.

Bearing-Only Approach.

In a similar vein to the bearing-range approach, Niculescu et al. [65] consider a system where nodes determine angles to their neighbors, and a subset of the nodes have global positioning capabilities, which is also a variant of triangulation. This work assumes positioning information is made available to some or all robots, my work does not make that assumption.

Bearing-only navigation has also been successfully implemented by Bekris with an omni-directional camera to localize to with angles to three landmarks. This framework combined with other techniques explores a variety of navigational tasks available with bearing-only coordinates [66, 67].

5.1.1 Communication Requirements

Bearing-only models are more limited than range-bearing models. For example range-bearing models need a small amount of communication to estimate the network geometry. However, the amount of inter-robot communication increases greatly when bearing-only models estimate the network geometry. The inter-robot communication requirement is often overlooked in the literature. However, algorithms that require large amounts of information from neighboring robots or many rounds of message passing are impractical on systems with limited bandwidth and high reliability. This work uses the bearing-only sensor model with scale-free coordinates to balance the trade-off between cost, complexity,

communications, and capability. Sensor type also plays a large role in the quality of the network geometry as discussed in vision-based systems [68] and infrared light [69, 70]. The r-one robot relies on IR communication for local network geometry estimation.

5.1.2 Mathematical Foundations of Scale-Free

From the theory literature, Whiteley [71] set the foundation for directional graph rigidity using the tools of matroid theory. This paper presents a less complex alternative algebraic characterization to directly compute scale-free coordinates based off Cornejo [62]. In addition, Bruck [72] addresses the problem of finding a planar spanner by only using local angles. The Bruck paper studies a similar problem of creating a a virtual coordinate system from local angles, but their focus delves into routing schemes for sensor networks.

5.2 Assumptions for algorithm

The following assumptions and concepts were used to compute scale-free coordinates.

5.2.1 k-Connectivity

In k communication rounds a robot u learns about other robots in G within a distance k. However, each robot learns about the edges between robots at distance $k-1$ and its k-neighborhood G_u^k. In addition to its k-neighborhood the robot also updates the bearings $\theta_u(w)$ and $\theta_v(w)$ for each edge $\{v, w\} \in E(G_u^k)$. We call this the labeled k-neighborhood of u and denote it with (G_u^k, θ).

5.2.2 Information Gathering

To gather the maximum amount of information at each round, the model runs a full-information protocol during the communication steps that make up a round. Full-information protocol involves each robot broadcasting a message containing all the information it knows

from one time step. This model of gathering information is specific to the hardware algorithm and not a general approach to computing scale-free coordinates.

Specifically, in the first communication step, each robot u broadcasts a message containing its unique identifier, and receives for every node $v \in N(u)$ a message with the identifier of v and records the bearing $\theta(u, v)$. In the second communication step, every robot u broadcasts its own identifier, along with the identifiers and bearings of each of its neighbors $v \in N(u)$. Therefore, by the end of the second communication step each robot u receives for each of its neighbors $v \in N(u)$ its identifier and the bearing $\theta(u, v)$, as well as the identifier of w and the bearing $\theta(v, w)$ for every node $w \in N(v)$.

In other words, by the end of a round, each robot u is aware of the identifiers of all the robots in its two-hop neighborhood $N^2(u)$, as well as the graph $G^1(u)$ together with the bearings $\theta(v, w)$ and $\theta(w, v)$ of every pair of nodes v and w which are neighbors in $G^1(u)$. We refer to the graph $G^1(u)$ together with the bearings $\theta(v, w)$ and $\theta(v, w)$ for every edge (v, w) in $G^1(u)$ as the labeled sub-graph of u.

5.3 Computing Local Scale-Free Coordinates

Computing local scale-free coordinates for robot u, finishes with finding a set of length assignments $\ell_u(v)$ for every robot $v \in N(u)$. If successful, the length ratios will match the ground truth physical length ratios between the robots. If there are multiple realizations of G_u^k with the same angle measurements, then it is impossible to compute scale-free coordinates. Thus, if (G_u^k, θ) has a *unique realization* when all the angle-satisfying realizations of (G_u^k, θ) have the same length ratios. A *unique realization* is synonymous with robot u being *rigid.* Therefore, Robot u can compute its local scale-free coordinates using (G_u^k, θ) if and only if robot u is rigid.

Given a realization p, ℓ_p denotes a *length vector* that assigns each edge $\{v, w\} \in E(G_u^k)$ of length $\|p(w) - p(v)\|$. Any realization p of G_u^k must satisfy for any directed cycle $\vec{C}$ in G_u^k the following equations, which correspond to the x and y components of a single

homogeneous linear two-dimensional vector equation,

$$\sum_{(w,v)\in E(\vec{C})} \ell_p(w,v)\cos(\theta_w(v)) = 0$$
$$\sum_{(w,v)\in E(\vec{C})} \ell_p(w,v)\sin(\theta_w(v)) = 0 \tag{5.1}$$

The matrix A is defined as the $2(m-n+1) \times m$ matrix that encodes the homogeneous system of linear equations that results from applying equation 5.1 to a *cycle basis* of G_u^k. A cycle basis of a graph is a set of simple undirected cycles present in the graph. Every connected graph with n vertices and m edges has a cycle basis with exactly $m-n+1$ cycles [73].

$$\begin{array}{c} \\ C_1^x \\ C_1^y \\ \vdots \\ C_q^x \\ C_q^y \end{array}
\underbrace{\begin{array}{c} \begin{array}{ccc} e_1 & \cdots & e_m \end{array} \\ \begin{bmatrix} & & \\ & & \\ & & \\ & & \\ & & \\ & & \end{bmatrix} \end{array}}_{A}
\underbrace{\begin{bmatrix} \ell_p(e_1) \\ \\ \vdots \\ \\ \ell_p(e_m) \end{bmatrix}}_{\mathbf{x}} = \mathbf{0}$$

p is a angle-satisfying realization of (G_u^k, θ) if and only if ℓ_p is in the null space of A.

Computing if ℓ_p is in the null space of A is a suitable procedure to compute the local scale-free coordinates of robot u. First, the labeled k-neighborhood of u is used to construct A, and the null space basis N of A is computed. Lengths of the edges (u,w) for $w \in N(u)$ are the primary interest of the algorithm and N can be reduced to only those edges to form matrix N'.

- When $N' = 1$, there exists a unique realization of (G_u^k, θ) and scale-free coordinates can be computed.
- Conversely when $N' > 1$, there exists multiple realizations of (G_u^k, θ) and scale-free coordinates can not be computed.

The computation of the null space, implemented using Singular Value Decomposition, dominates the complexity of this procedure, and therefore the total complexity is $O(m^3)$ where m is the number of edges in G_u^k.

The computation procedure begins with each robot collecting enough information to recover its k-neighborhood, which requires $k+1$ communication rounds. If the maximum degree of G_u^k is Δ, then all messages are of size at most $O(\Delta^k)$. Each robot must receive Δ messages and transmit one, so the required bandwidth measured as bits/robot/round will be $O(\Delta^k)$.

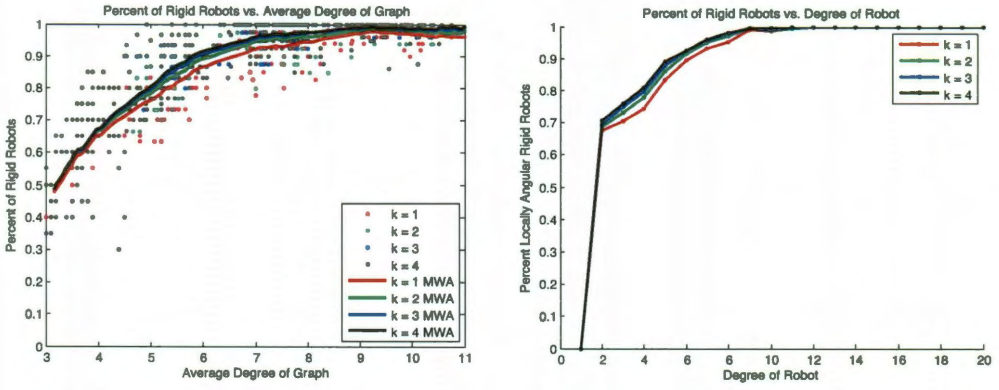
5.4 Simulation Results

The simulation considers a circular environment with a 4 m diameter and 10 cm diameter robot. The simulation assumes lossless bidirectional communication and noiseless bearing-only sensors with a $1m$ communication interface. The parameters in the experiments are the communication depth k and the population size. When running the simulation each robot u first recovers its labeled k-neighborhood (G_u^k, θ). Finally, the robots run the previously described procedure to compute their local scale-free coordinates, or determine that the local scale-free coordinates cannot be computed.

5.4.1 Simulation Details

Simulations were performed using populations of 20, 30, 40, and 50 robots, running 100 experiments for each population size. Of these 400 experiments, 43% of these graphs had a unique realization and successfully computed local scale-free coordinates.

The results shown in Figure 5.1(a) are encouraging. Typical graphs with an average degree of 6-7 produce around 90% rigid robots even with a communication depth of only $k = 1$. Low average degree graphs of 4 are still able to have at least 50% of the robots compute their local scale-free coordinates. For graphs of degree 6, approximately 80% of the robots can compute their local scale-free coordinates using $k = 1$, which suggest it is



(a) Rigidity vs. Degree, Moving Window.

(b) Rigidity vs Degree.

Figure 5.1: **a:** Percentage of rigid robots versus average degree of graph for varying communication depths in all generated graphs. A moving window average for each communication depth is overlaid on the plot. **b:** Percentage of rigid robots versus degree of robot for varying communication depths in all generated graphs. Both graphs were originally presented in Cornejo, Lynch, Fudge, Bielstein and McLurkin [74].

feasible to compute local-scale free coordinates on bandwidth-limited platforms.

Figure 5.1(a) for $k = 1$ also allows for a direct comparison to the robust quad results of Moore [38]. The localization success rates are somewhat better than robust quad results, with the same communication depth $k = 1$, around 10% more nodes with low degree localize with bearing-only measurements than the robust-quads algorithm with distance-only measurements.

In addition, increasing communication depth from $k = 1$ to $k = 2$ increases likelihood of a given robot being rigid. Subsequent increases in k have diminishing returns. For example, Figure 5.1(b) shows that increasing communication depth never decreases the percentage of robots which are rigid.

5.5 2-Hop Scale-Free Algorithm

The simulation data presented utilized Singular Value Decomposition to compute the null space of the system of cycle equations. However, many low-cost robots have limited processing power and no floating-point unit, which make this an unrealistic approach. This section describes a less computational algorithm to compute local scale-free coordinates. The approach is constrained to small neighborhoods of robots, $G^1(u)$. Based on our simulation results, this is an acceptable compromise to implement local scale-free coordinates in multi-robot systems.

5.5.1 Assumptions and Criteria

Only cycles of three robots are considered in this algorithm. In other words, each robot u has two neighbors and forms a sub-graph of $G^1(u)$. The bearing measurements for each robot u are assumed to be exact. In the physical robots, this assumption is handled with a heuristic for noisy sensor measurements. Computing scale-free coordinates is possible if and only if a robot u can be removed from the sub-graph $G^1(u)$ and still be a connected graph. A proof for this claim is provided in previous work by Cornejo [62, 74].

5.5.2 2-Hop Scale-Free Psuedo-code

The following 2-Hop Scale-Free algorithm computes local scale-free coordinates of a robot.

This procedure has a running time which is linear in the number of bearing measurements in G^1_u. The SineLaw function takes u, z and w as parameters to form a edge length (u, v) of a u, z, w triangle. The length ℓ_z of the edge (u, z) is known and the angles of the u, z, w triangle are represented as θ. To return the length of the edge (u, v) it performs the following computation:

Algorithm 1 2-HOP SCALE-FREE algorithm running at node u

mark $v \in N(u)$ as WHITE, where v is a neighbor of u
mark the first v as BLACK and set $\ell_v \leftarrow 1$
$Q \leftarrow queue(v)$
while $Q \neq \varnothing$ **do**
    $z \leftarrow Q.pop()$
    for each WHITE $w \in N(z) \cap N(u)$ **do**
        mark w as BLACK and set $\ell_w \leftarrow$ SINELAW (u, z, w)
        $Q.push(w)$
    end for
end while

$$\text{SINELAW}(u, z, w) = \ell_z \left| \frac{\sin(\theta(z, u) - \theta(z, w))}{\sin(\theta(w, z) - \theta(w, u))} \right|$$

5.5.3 Noise Sensitivity

Sensor errors and limited precision are present in any physical system. Due to these inaccuracies the inner angles of each triangle will not always sum to π. This section discusses a method to cope with summing to π errors and reduce errors in the computation of local scale-free coordinates.

To ensure the angles sum to π an ANGLEBALANCE computation is applied to the inter-angles of the triangle as shown below. This procedure has a running time which is linear in the number of bearing measurements. The fundamental idea behind the 2-HOP SCALE-FREE algorithm is to traverse a tree of triangles. The lengths of each each triangle (u, z, w) are computed using the SINELAW procedure. In turn, the SINELAW procedure uses the bearing measurements to compute two inner angles ψ_z and ψ_w of the triangle to return a length proportional to $\sin(\psi_z)/\sin(\psi_w)$. When $\psi_z = 0$ or $\psi_w = 0$ then the triangle is degenerate then a length will not be computed.

Algorithm 2 ANGLEBALANCE takes input of TriList

1: **for each** $triangle \in TriList$ **do**

2: $SumDiff = \pi - (\phi(u), \phi(v), \phi(w))$

3: $\phi(u) = \phi(u) + \frac{SumDiff}{3}$

4: $\phi(v) = \phi(v) + \frac{SumDiff}{3}$

5: $\phi(w) = \phi(w) + \frac{SumDiff}{3}$

6: **end for**

Therefore, the angles ψ_z and ψ_w can be used to characterize the *noise sensitivity* of each triangle (u, z, w). The noise sensitivity of a triangle represents the expected error on the length computed using that triangle. For example, as ψ_z gets closer to zero, a triangle becomes more sensitive to noise, since a small change in the bearings used to compute ψ_z translate to large changes in the computed length. To be precise, the vector gradient of $\sin(\psi_z)/\sin(\psi_w)$ can be used to characterize the noise sensitivity of a triangle (u, z, w).

To reduce the effect of noise in the sensing measurements, instead of traversing an arbitrary tree, a minimal spanning tree is used with noise sensitivities as the weights. This minimizes the total noise sensitivity of the triangles used. The vector gradient used to obtain the weights is described in the SINCOS computation below.

Algorithm 3 SINCOS weighting for each triangle connected to node u

1: **for each** $triangle \in TriList$ **do**

2: $SinVW = \frac{sin(\phi(v))}{sin(\phi(w))}$

3: $CosSinVW = (\frac{cos(\phi(v))}{sin(\phi(w))})^2 + (\frac{sin(\phi(v)) * cos(\phi(w))}{sin^2(\phi(w))})^2$

4: $weight = (\frac{SinVW}{(CosSinVW + SinVW) - 1})^2$

5: **end for**

5.5.4 2-Hop Scale-Free Psuedo-code with Noise Sensitivity

The complete algorithm with angle correction and noise sensitivity is shown below.

The 2-HOP SCALE-FREE algorithm with noise sensitivity works effectively in simu-

Algorithm 4 2-HOP SCALE-FREE algorithm running at node u with ANGLEBALANCE and SINCOS noise sensitivity.

execute ANGLEBALANCE on all triangles in $G^1(u)$
compute all weights of triangles in $G^1(u)$ using SINCOS
form Minimum Spanning Tree (MST) using weights
mark all $v \in N(u)$ as WHITE, where v is a neighbor of u
mark the first v in MST as BLACK and set $\ell_v \leftarrow 1$
$Q \leftarrow queue(v)$
while $Q \neq \varnothing$ **do**
    $z \leftarrow Q.pop()$
    for each WHITE $w \in N(z) \cap N(u)$ **do**
        mark w as BLACK and set $\ell_w \leftarrow$ SINELAW (u, z, w)
        $Q.push(w)$
    end for
end while

lation. However, physical experiments are an important step to validating the practical aspects of the algorithm. The hardware experiments in the next chapter will demonstrate the effectiveness of the 2-HOP SCALE-FREE algorithm in centroid-seeking behaviors.

5.6 Relative Power of Sensing

In terms of sensors, this work is a small part of a much larger discussion about the complexity and type of sensors required to perform a particular task as seen from Donald, Erdmann [75, 76]. O'Kane later formulated there is no consensus in the literature of how to best understand and quantify the relative "power" of robots and sensors [77]. The general themes are constructing and ordering based on sensor capability which solves particular tasks more directly or efficiently.

5.6.1 Dominance Relations

O'Kane introduces the idea of a *dominance* of one robot model over another.

> A dominance relation is defined as a robot R_2 *dominates* another robot R_1 if R_2 can collect at least as much information as another robot R_1.

To classify which robot dominates, the term *robotic primitive* is used to describe the sensing and motion capabilities of a particular robot. For example, a robot measuring linear distances and orientation dominates a robot that only measures linear distances.

This work is interested in stratifying dominance relations between coordinate systems. Each coordinate system has an associated set of robotic primitives such as range-only or bearing-only sensing. This work extends the dominance relation of coordinate systems from Rykowski [52]. Specifically, the coordinate system dominance relation classifies scale-free coordinates between global/local and bearing/range-only coordinate systems as shown in Figure 5.2. Global coordinates given by devices like GPS is the most dominant coordinate system. *Local coordinates* provide relative pose (x, y, θ) of neighboring robot positions.

Computing (x, y) pose of a robot consists of a robot converting range and bearing to (x, y) coordinates. The range and bearing measurements are dependent on a variety of local sensing on the robot. Bearing-only coordinates are more dominant than range-only coordinates due to the geometric information provided by bearings of neighbors. Estimating the shape of the network becomes more difficult with only range measurements. With only bearing and a known observation model, a robot can use *probabilistic range estimation* to obtain virtual range information. Probabilistic range estimated coordinates are possible with scale-free and a particle filter technique by Rykowski [52]. Probabilistic range estimation methods combine information from the robot sensing model, robot odometry and a robot motion model. Rykowski's technique assumes the motion profile of the robot is broadcasts to other robots estimating range. Scale-free coordinates only use network geometry and does not rely on a robot motion model as an input. Therefore, scale-free coordinates are disjoint in some respects from particle filter estimation techniques. Comparing probabilistic range estimation to scale-free coordinates is not a clear comparison without assumptions about a particular network. For example, the Rykowski particle filter approach combines sensor error models and odometry motion models to predict the most likely position of the neighbor robot. While scale-free coordinates does not rely on odometry and requires a network of three or more robots. A more careful comparison between similar estimation techniques and robotic primitives is needed to clarify the differences.

The scale-free approach is a geometric technique to produce an approximate estimate of range between robots up to a scaling-factor. Scale-free coordinates also requires a network of robots to form a cycle with angular measurements of the cycle inner angles. Since scale-free coordinates are not possible in a two-robot network, probabilistic range estimation is the more dominant coordinate system for two robots. With a network of three or more robots, scale-free coordinates will provide a range estimate between robots up to an unknown scale-factor. A combination of range estimation and scale-free coordinates has the potential to be the most dominant coordinate system of the three. However, a combined approach will be taxing on the robot's computation and communication requirements.

Therefore, an efficient algorithm combining scale-free and particle filter estimation shows a promising future for multi-robot systems.

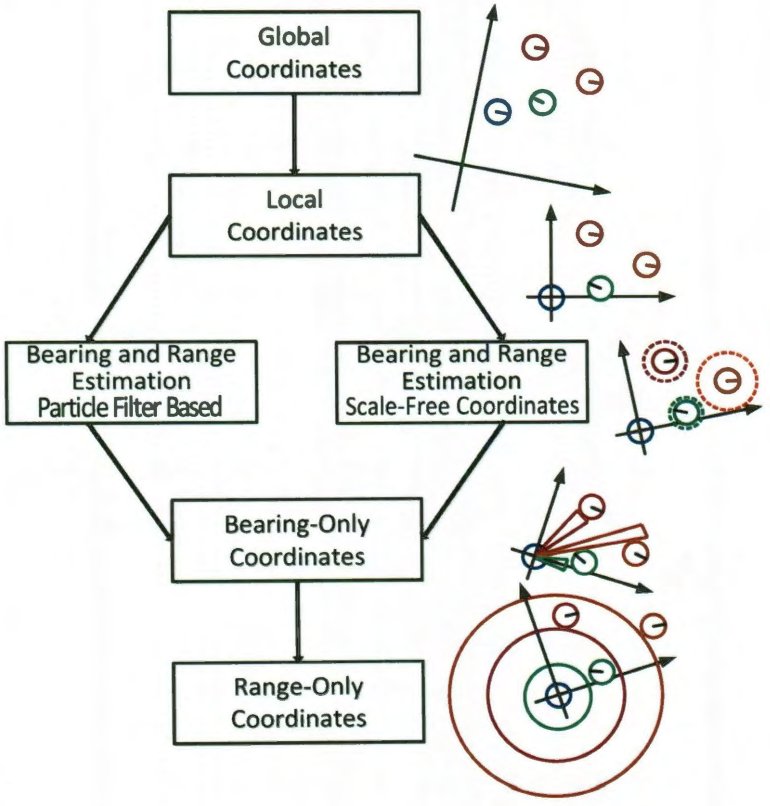


Figure 5.2: Dominance Relation between Coordinate Systems. The column on the left represents the hierarchy of the coordinate systems with a corresponding coordinate illustration on the right from Rykowski [52].

Chapter 6

Hardware Experiments

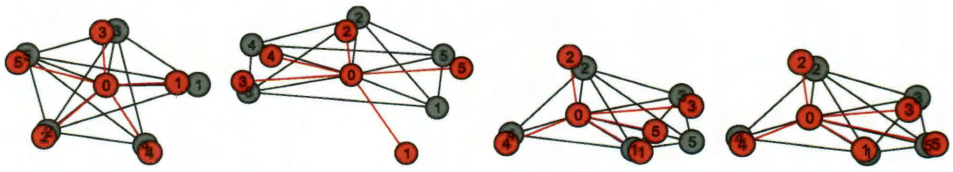
In this chapter, the 2-Hop Scale-Free algorithm is implemented on the r-one robots. The robot experiments validate the practical attributes of this algorithm in static configurations and motion behaviors.

6.1 Static Evaluation

This work first evaluates this algorithm on stationary configurations. However, the results presented are applicable while the system is in motion as long as the physical speed of the robots is negligible compared to the speed of communication and computation [19]. Random connected geometric graphs were generated by placing six robots uniformly around the environment and discarding disconnected graphs.

The static evaluations include 32 random configurations of six r-one robots. Limiting the number robots at six is important for two reasons. First, a single robot has a limit of eight neighbors over infrared communication and six provides a comfortable margin from this maximum to avoid communication collisions. Six is also a ideal number for 2-hop networks in the r-one's communication diameter as noted by Klienrock [78]. Specifically, six robots is the proper balance for a network using an ALOHA slotted access scheme with randomly positioned nodes. The trade-off between communication radius and average degree connectivity between the network shows a degree of six offers a suitable network bandwidth capacity in large networks.

In the evaluations, the robots use the 2-Hop Scale-Free algorithm to recover their local scale-free coordinates. 4 of the 32 trials failed due to lost messages between robots,



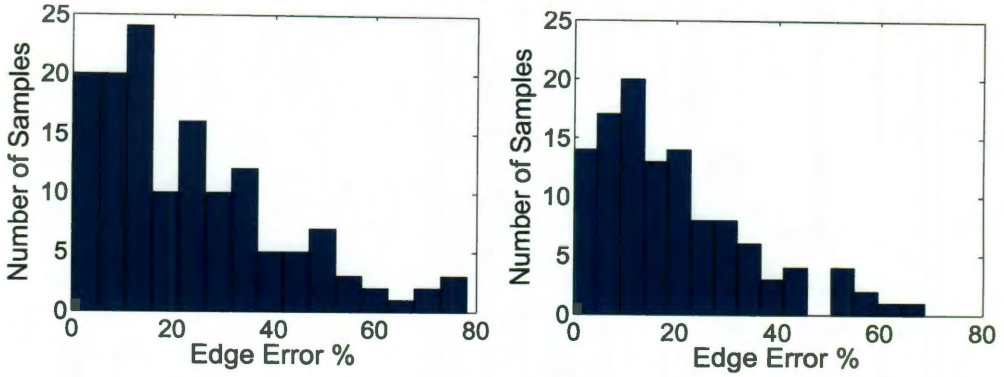
(a) successful case. (b) bearing error. (c) edge error. (d) error corrected.

Figure 6.1: Scale-free coordinates plotted as red nodes and ground-truth data as grey nodes. The IR communication links are plotted as black edges between grey nodes. The red lines depict the measured bearing between each robot. Four cases are presented: **a:** Accurate scale-free coordinates. **b:** Configuration with bearing errors on robot 1. **c:** Scale-free edge error on robot 5. **d:** Scale-free robot 5 edge corrected with noise sensitivity.

those four were discarded and the remaining 28 successful trials were then analyzed. A sample of these configurations are shown in Figures 6.1(a)- 6.1(d) to illustrate the performance and errors observed in the experiments. The figures display the ground-truth camera data as grey nodes and the algorithm's scale-free coordinates as red nodes. The IR communication links between the robots are shown as black edges. The actual bearing measurements from the central robot to each of its neighbors are depicted by red lines. Ideally, the red nodes and edges will directly cover the black edges and grey nodes. However, low resolution sensor measurements will prevent scale-free coordinates from exactly matching with physical coordinates. The low resolution case was the most frequent source of error and produced the most significant edge errors. Other sources of error include lost messages between robots typically resulting from collisions over the infrared medium. The last error seen is a noise sensitivity computation error caused by picking the non-optimal set of triangle in the algorithm. This step was later improved in the algorithm with a heuristic but the results of noise sensitivity are still presented.

For each static configuration, a unique α scaling factor was computed to minimize potential edge errors. The α was choosen based on ground-truth camera information to show the best possible performance of scale-free coordinates. Minimizing edge errors for static configurations provides a best estimate of the mean error in scale-free calculations of the robots. Optimizing α was suitable for analyzing the errors in scale-free coordinates. However, picking a unique α is not suitable for real-time hardware because robots do not have access to ground-truth data.

An example of a low-resolution bearing measurement is shown in Figure 6.1(b) for robot 0 to robot 1. Despite this error, the 2-HOP SCALE-FREE algorithm still effectively computes the edge coefficient. The majority of the low-resolution errors are still within the 22.5°designed tolerance of the robot. Low-resolution sensors and lost messaging errors cause the most significant problems in the robot's scale-free coordinates. Three of the 31 one experiments had lost messages which resulted in not being able to compute scale-free coordinates. The other errors present in the histogram were caused by low-resolution sensing and noise sensitive computations. Figure 6.1(c) demonstrates scale-free edge errors with robot 5. The bearing of robot 5 is correct yet the scale-free edge length is inaccurate. In this case, the error was caused by a poor selection of triangles. Selecting the best triangles can be improved with the noise sensitivity heuristic. The noise sensitivity heuristic corrected the position of robot 5 from Figure 6.1(c) to more accurate edge coefficient as shown in Figure 6.1(d).



(a) Edge Error.

(b) with Noise Sensitivity.

Figure 6.2: **a:** Edge error histogram for 28 robot configurations. There are total of 140 edges in this data set. This histogram has a mean percent error of 23.4%. **b:** A corresponding edge error histogram with noise sensitivity added on the same data set as the left. This histogram reduces the mean percent error to 19.4%.

For the 28 experiments, 140 edges were run using scale-free coordinates, a histogram of the edge error percentage is shown in Figures 6.2(a) - 6.2(b). Running the algorithm without noise sensitivity produces a mean percent error of 23.4% as shown in Figures 6.2(a). However, when the same data set is run with the noise sensitivity heuristic, the mean percent error reduces to 19.4% as shown in Figure 6.2(b). Given the coarse bearing measurements from the r-one, these results are reasonable and still usable for motion control.

6.2 Dynamic Evaluation: Centroid Behavior

This work uses a centroid-seeking behavior to validate the 2-HOP SCALE-FREE algorithm. The moving robot will converge on the centroid of a group of robots.

6.2.1 Related Centroid Experiments

This work searched for related approaches of minimalistic sensing for centroid convergence with a group of robots. Yu et al. demonstrates a simulation of converging robot agents with a simple control law [79]. This work assumes agents will converge into each other but do not have bearing or range information. The convergence proof presented demonstrates the viability of clustering between a group of robots with minimal sensing and no multi-robot communication. This work is different than Yu et al. by focusing on one robot reaching the center of a group of robots. This centroid seeking behavior will stabilize a group of robots into a desired shape. This centroid behavior also scales well with large populations of robots over large areas. Covering an area with a group of robots is referred to as *distributed coverage*. Schwager and McLurkin addressed the problem of distributed coverage with experiments demonstrating 16 SwarmBots performing voronoi centroid coverage of an area [80]. These robots had local pose information with range-bearing measurements and used consensus over distributed communication to stabilize the network. This work differs by investigating centroid behaviors with bearing-only information and scale-free coordinates.

6.2.2 Centroid Experiments

This work presents a controller to navigate to the centroid of a group of robots. The moving robot uses the most recent calculation of scale-free coefficients and does not average sensor measurements over time. Independence from time enables the experiment to validate the true sensor performance of the system without filtering or averaging.

Due to long 2.5 *sec* neighbor rounds, measuring neighbor bearings while moving can

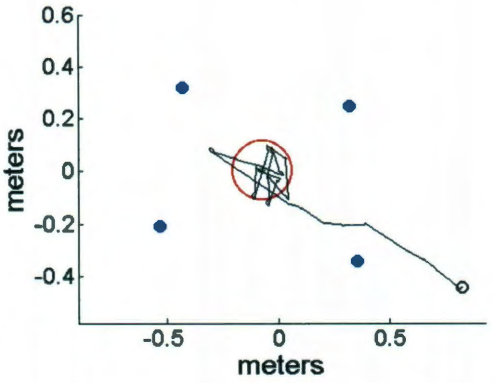
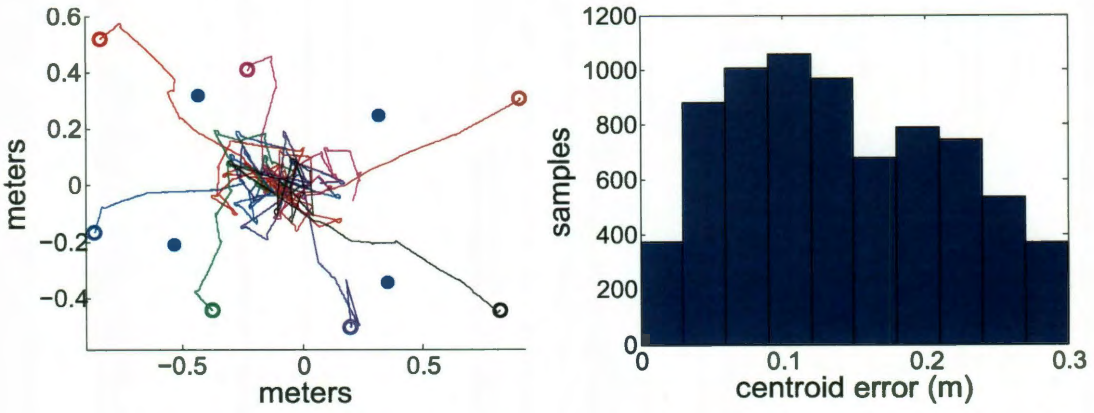


Figure 6.3: Four static robots shown as blue dots were placed in an arbitrary polygon. The motion robot trajectory is the black line starting from the initial black circle. The motion robot uses the 2-HOP SCALE-FREE algorithm to compute local scale-free coordinates and the a local centroid estimate. The robot converges within the red circle of radius step distance of $d_{step} = 11cm$.

introduce errors between measurements taken at different times. For these experiments, the robot pauses in a stationary state when measuring neighbor bearings. The bearings and communication network are combined to produce scale-free coordinates. The robot then computes an angle to the scale-free centroid. The robot moves toward that angle at a fixed distance of $d_{step} = 11cm$. After one iteration of the algorithm, the robot pauses for the next neighbor round and repeats the process. The motion profile of the robot is not fluid because the limited communication bandwidth forces the robot to stop between neighbor rounds. For the first experiment, four stationary robots were arranged in an arbitrary polygon and one moving robot outside the polygon. An illustration of the centroid trajectory is shown in Figure 6.3. The robot's trajectory moves around the centroid without settling. This behavior is expected because the robot has no notion of exact distance to the centroid, thus, the robot cannot stop and will continue circling. The robot is expected to stay within the radius d_{step} of the red circle denoted in the figure.

The centroid experiment was extended for many initial starting conditions. The trajectories of the moving robot converging to the centroid is shown in Figure 6.4(a). With



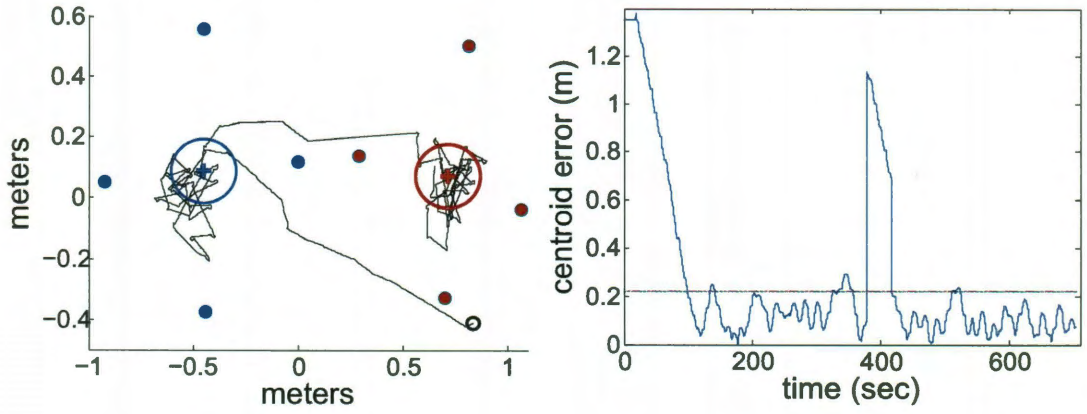
(a) Centroid convergence.

(b) Convergence error.

Figure 6.4: Motion Control Experiment - **a:** Four static robots shown as blue dots were placed in an arbitrary polygon. The motion robot was placed in random locations shown as colored circles outside the polygon. Convergence trajectories of the motion robot moving toward a centroid are shown by the different colored lines. The motion robot uses the 2-HOP SCALE-FREE algorithm to compute local scale-free coordinates. **b:** Corresponding error histogram between motion robot position and the centroid from the different trajectories shown in (a). The errors outside the polygon are not included to demonstrate the error inside the polygon. The robot oscillates around the centroid as a function of the maximum step distance of $d_{step} = 11cm$. The mean error of this plot is 14.03 cm.

a larger data set, the diameter of the convergence region does not always describe the motion profile of the robot trajectories. However, a histogram of robot distance to the centroid shown in Figure 6.4(b) provides a mean error of 14.03 cm which is well within the $2d_{step} = 22cm$ convergence circle diameter.

The second centroid experiment shows the moving robot tracking the stationary robots in two different positions. The stationary robots start in the blue positions, then were shifted to the red positions. The trajectory shown in Figure 6.5(a) show the moving robot successfully converging to the new position, and the size of the convergence region in Figure 6.5(b) is within d_{step} radius of the convergence circle.



(a) Centroid shift.

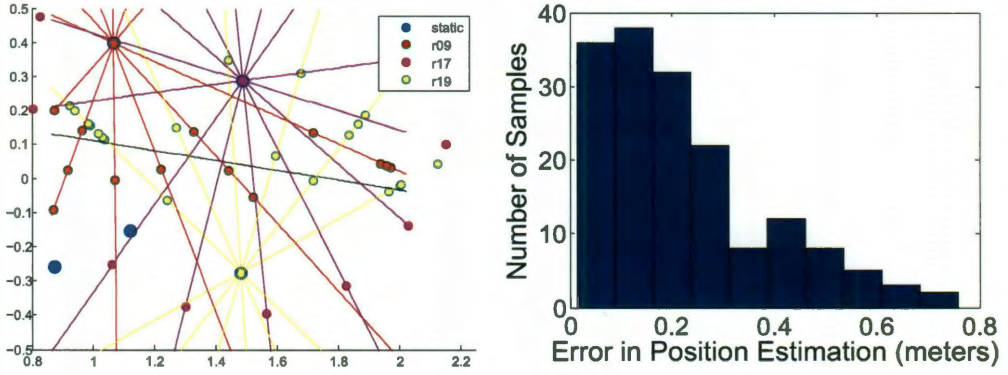
(b) Centroid shift error.

Figure 6.5: **a:** This experiment moves a group of robots to demonstrate a large shift in the centroid denoted by the blue plus sign. The four blue dots are the initial polygon of static robots. The red dots represent the shifted group of robots. The black line trajectory shows the trajectory of the motion robot searching for the centroid. The red and blue circles represent the convergence of a fixed step size with a radius of $11cm$. The robot is expected to oscillate within this circle. **b:** Corresponding error vs. time of the trajectory shown in Sub-figure (a) between the motion robot position and the centroid. The robot begins at the black circle with significant error and then oscillates less than d_{step} radius around centroid. When the group is shifted the error spikes again and settles to another oscillation around the new centroid.

6.3 Dynamic Evaluation: Tracking Motion

This experiment set out to track motion trajectory of the moving robot using scale-free coordinates on the stationary robots. Analyzing scale-free coordinates between multiple robots increases the volume of data to process. The experiment consisted of five stationary robots in a connected graph configuration. A motion robot traversed this network with a pre-defined straight line motion. The stationary robots produced an estimated position of the motion robot with with scale-free coordinates and a α scaling factor. When combined together at each time instance, this trajectory provides a reasonable estimate of the motion

robot trajectory as shown in Figure 6.6(a). The estimates produced by the robots only fall along the bearing lines from each stationary robot drawn as the red, blue or yellow lines. The errors from the experiment are plotted in Figure 6.6(b).



(a) Tracking Motion Robot.

(b) Total Error Histogram.

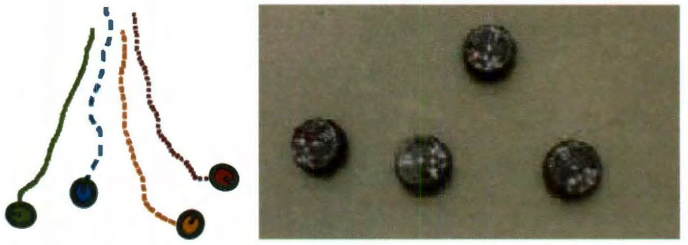
Figure 6.6: **a:**The blue static points represent the observer positions. The black path is the motion path of the single moving robot. The colored dots are estimates of the moving robots position from scale-free coordinates. **b:** Combined Histograms of error in meters from the bearing line intersection with the moving black line. The robots only see robots along their respective bearing lines.

The tracking experiment is interested in the possibilities with multi-robot scale-free behaviors. For now the experiment mainly focused on the performance of the sensors and 2-HOP SCALE-FREE algorithm under realistic network conditions. In future work, the centroid seeking and trajectory tracking can be combined to perform a centroidal vornoi network.

6.4 Multi-Robot Behaviors

Scale-free coordinates enhances bearing-only sensors and behaviors with the additional network communication. This work is also interested in the potential of bearing-only multi-robot behaviors. Enhancements to these bearing-only behaviors with scale-free coordinates are possible later in the future. For now, I present a series of bearing-only behaviors successfully implemented on the r-one.

6.4.1 Flocking



(a) Flocking Behavior.

(b) Flocking Experiment.

Figure 6.7: **a:**Illustration of flocking behavior in which each robot copies the orientations of it's neighboring robots. The robots will move forward at a fixed speed and only turn when a obstacle is detected with the bump sensors. **b:**Flocking experiment with the r-ones. These robots have only bearing and orientation information and do not have range between robots.

Bearing-only flocking is only possible with knowledge of orientation of neighbors. The r-one is capable of *angle-only* flocking behavior in simple environments. In this example a flock of r-ones flock in a rectangular environment. Imagine a flock of birds forming a flying v formation, in this case the r-ones will not form a specific shape but will flock together in a general direction. The flocking behavior consists of robots matching all neighbor orientations while continually moving forward. Each robot's individual path is plotted in a unique color in Figure 6.7(a). The flock is able to handle walls and simple obstacles.

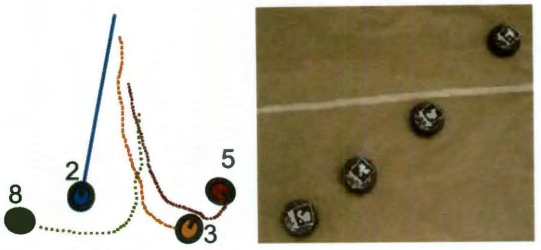
The experiment in Figure 6.7(b) demonstrates that angle-only flocking works properly with four robots.

The angle-only flocking approach is not equivalent to range-bearing flocking but provides a promising example of the r-one capabilities.

Algorithm 5 Flocking algorithm running at robot u

1: **while** 1 **do**
2: get $\psi(v) \in N(u)$, update orientations, where v is a neighbor of u
3: $\psi_{avg} = average(\psi(v) \in N(u))$, Average neighbor orientations
4: PoseSet(tv = 20 mm/sec, $heading = \psi_{avg}$), set pose of robot u
5: **if** CheckBumpObstacle(u) **then**
6: PoseSet(tv = 0 mm/sec, $heading = \theta_u + \pi$), turn away from obstacle
7: **end if**
8: **end while**

6.4.2 Sorted Following



(a) Sorted-Follow Behavior.

(b) Sorted-Follow Experiment.

Figure 6.8: **a:**Illustration of sorted-order following behavior in which the lowest ID robot in the network avoids obstacles with infrared. The second lowest ID robot will follow the lowest ID robot in the network. This pattern continues in sorted order by robot IDs. **b:**Sorted-follow experiment with the r-ones.

This behavior assumes each robot in the network has a unique identifier or *ID*. The lowest numbered ID in a network of robot will perform an obstacle avoidance behavior with infrared sensors. The second lowest ID robot will follow the lowest ID robot in the network. With only two robots, the behavior exhibits basic follow-the-leader characteristics. While following, the robots will avoid obstacles only with bump sensors. The algorithm scales by inserting more robots with higher IDs to follow in sorted order. If the leader of the line is removed, the next lowest ID becomes the leader. Following sorted order will scale for multiple robots and typically works best with a group of 3-6 robots. Figure 6.8(a) depicts each robots sample paths while performing the sorted-order follow behavior. The blue robot is moving straight ahead in obstacle avoidance mode but not explicitly following any other robot. Sorted-order following is a distributed algorithm, thus the blue robot can be removed and the next lowest ID will take the place of the lowest ID robot. This algorithm is practical for navigating groups of robots through narrow passage-ways. Figure 6.8(b) illustrates sorted-order following behavior with the four r-one robots.

Algorithm 6 Sorted Following algorithm running at robot u

```
1: while 1 do
2:     LowestID=True
3:     get θ(v), id(v) ∈ N(u), update bearings and IDs of v neighbors
4:     for each id(v) ∈ N(u) do
5:         if id(v) < id(u) then
6:             PoseSet(tv = 20 mm/sec, heading = θ(v))), follow neighbor v
7:             if CheckBumpObstacle(u) then
8:                 PoseSet(tv = 0, heading = θ_u + π), turn away from obstacle
9:             end if
10:            LowestID=False
11:            break
12:        end if
13:    end for
14:    if LowestID==True then
15:        PoseSet(tv = 20 mm/sec, heading = 0)
16:        AvoidIRObstacle(u), the leader of the line
17:    end if
18: end while
```

6.4.3 Clustering

(a) Cluster Behavior. (b) Cluster Experiment.

Figure 6.9: **a:**Illustration of cluster behavior in which the even ID robots move toward each other forming a cluster. The odd ID robots do the same behavior with only odd robots. Clustering will not reach a stopping condition because r-ones do not have range information. **b:**Clustering experiment with four r-one robots.

This approach to clustering creates a cluster of robots based on even or odd robot IDs. Clustering on the r-one does not use range information and can be supplemented with bump information. As the robots collide with their respective cluster, they use the bump sensor to detect this event. Trajectories with 4 sample robots are shown in Figure 6.9(a). The figure demonstrates how robots will move toward their respective even or odd cluster. To allow for additions to the network, the robots never stop in the cluster. A screenshot of four r-one robots clustering is show in Figure 6.9(b) Similar to a swarm of insects the robots will continue to collide together in a cluster. Clustering with angle-only flocking has the potential to enhance flocking on the r-one.

Algorithm 7 Cluster algorithm running at robot u

1: **while** 1 **do**
2: $FollowMode$ = False
3: get $\theta(v), id(v) \in N(u)$, update bearings and IDs of v neighbors
4: **for each** $id(v) \in N(u)$ **do**
5: **if** $EvenOrOdd(id(v)) == EvenOrOdd(id(u))$ **then**
6: $FollowMode$ = True
7: $\theta_{avg} = average(\theta(v), \theta_{avg})$, Average neighbor bearings
8: **end if**
9: **end for**
10: **if** $FollowMode$ **then**
11: PoseSet(tv = 20 mm/sec, $heading = \theta_{avg}$)), follow even or odd neighbors
12: **if** CheckBumpObstacle(u) **then**
13: PoseSet($tv = 0$, $heading = \theta_u + \pi$), turn away from obstacle
14: **end if**
15: **end if**
16: **end while**

6.4.4 Mid-angle Navigation

Mid-angle navigation is another interesting problem solved with bearing-only information. This behavior includes a seeker motion robot moving toward a goal robot through a network of robots. The seeker robot typically does not have direct communication with the goal robot. However, the seeker communicates with the network of robots to determine the best path to approach the goal. To navigate, the seeker robot moves through the mid-angle bearing measurements of the network to avoid collisions. The trajectory of the seeker robot through the network is shown in Figure 6.10. This behavior is practical for navigating through large populations of robots to a desired goal position. For example, a seeker robot low on battery needs to return back to the charging station. The closest robot to the charging station becomes the goal and the low battery robot will quickly route through the network of robots to regain charge.

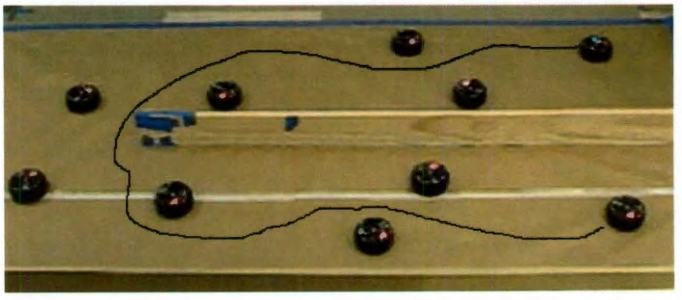


Figure 6.10: Illustration of mid-angle navigation through a corridor with a sample trajectory drawn in black. The goal robot is highlighted in green. This is an approximation of the algorithms performance not a ground-truth measurement.

Algorithm 8 Midpoint Navigation algorithm running at goal seeking robot u, all other robots in network are stationary and broadcasting their hop distance from the goal robot.

1: **while** 1 **do**
2: $FollowGoalRobot, FollowMidGoal$ = False
3: get $\theta(v), id(v) \in N(u)$, update bearings and IDs of v neighbors
4: **for each** $id(v) \in N(u)$ **do**
5: **if** $GoalRobot(id(v))$ **then** , Goal Broadcasts IR message to all robots in range
6: $FollowGoalRobot$ = True, $\theta_{goal} = \theta(v)$
7: **end if**
8: **end for**
9: **if** $FollowGoalRobot$ **then**
10: PoseSet(tv = 20 mm/sec, $heading = \theta_{goal}$)), follow goal
11: **if** CheckBumpGoal(u) **then** break out of loop, goal is found
12: **end if**
13: **end if**
14: **if** !$FollowGoalRobot$ **then**
15: **for each** $id(v), id(w) \in N(u)$ **do**
16: **if** $LowestHopsToGoal((id(v)) == LowestHopsToGoal(id(w))$ **then**
17: $FollowMidGoal$ = True, $\theta_{midgoal} = average(\theta(v), \theta_(w))$,
18: **end if**
19: **end for**
20: **end if**
21: **if** $FollowMidGoal$ **then**
22: PoseSet(tv=20mm/sec,$heading = \theta_{midgoal}$)), avoid obstacles
23: **end if**
24: **end while**

Chapter 7

Conclusion

This thesis presented a low-cost multi-robot system, a new scale-free coordinate system and a series of experiments to validate the coordinate system with the robot. The *r-one* robot platform is suitable for research, education and outreach. The rone's *angle-only* sensor model is effective for multi-robot behaviors such as clustering or sorted following. Scale-free coordinates along with the 2-HOP SCALE-FREE algorithm sets a foundation for localizing with large populations of simple, low-cost robots. The scale-free algorithm is tailored to low-cost systems with limited communication bandwidth and sensor resolution. The algorithm also uses a noise sensitivity model to reduce the impact of noise on the computed scale-free coordinates. The r-one combined with scale-free coordinates have set a foundation for future distributed algorithm research in multi-robot systems.

Bibliography

[1] M. J. Mataric, "A distributed model for mobile robot environment-learning and navigation," tech. rep., Cambridge, MA, USA, 1990.

[2] J. Forlizzi and C. DiSalvo, "Service robots in the domestic environment: a study of the roomba vacuum in the home," in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, HRI '06, (New York, NY, USA), pp. 258–265, ACM, 2006.

[3] B. Yenne, *Attack of the drones: a history of unmanned aerial combat.* MBI Pub. Co., 2004.

[4] C. Urmson, C. Baker , J. Dolan, P. Rybski, B. Salesky, W. R. Whittaker, D. Ferguson, and M. Darms, "Autonomous driving in traffic: Boss and the urban challenge," *AI Magazine*, vol. 30, pp. 17–29, June 2009.

[5] J. McLurkin, A. Lynch, S. Rixner, T. Barr, A. Chou, K. Foster, and S. Bilstein, "A low-cost multi-robot system for research, teaching, and outreach," *Proc. of the Tenth Int. Symp. on Distributed Autonomous Robotic Systems DARS-10, November*, p. 200, 2010.

[6] M. Batalin and G. S. Sukhatme, "Spreading out: A local approach to multi-robot coverage," in *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems*, (Fukuoka, Japan), pp. 373–382, June 2002.

[7] J. McLurkin and E. D. Demaine, "A distributed boundary detection algorithm for Multi-Robot systems," in *Proceedings of IEEE Intelligent RObots and Systems (IROS)*, (St. Louis, MO), 2009.

[8] T. S. Dahl, M. J. Mataric, and G. S. Sukhatme, "Emergent robot differentiation in distributed Multi-Robot task allocation," in *7th International Symposium on Distributed Autonomous Robotic Systems (DARS'04)*, pp. 191–200, 2004.

[9] A. Howard, M. J. Matarić, and G. S. Sukhatme, "Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem," in *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems*, p. 299—308, 2002.

[10] R. Arkin and T. Balch, *Line-of-Sight Constrained Exploration for Reactive Multiagent Robotic Teams.* 2002.

[11] R. Wei, R. Mahony, and D. Austin, "A bearing-only control law for stable docking of unicycles," in *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 4, pp. 3793–3798 vol.3, 2003.

[12] T. Lemaire, S. Lacroix, and J. Sola, "A practical 3D bearing-only SLAM algorithm," in *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pp. 2449–2454, 2005.

[13] S. Scheding, G. Dissanayake, E. Nebot, and H. Durrant-Whyte, "An experiment in autonomous navigation of an underground mining vehicle," *Robotics and Automation, IEEE Transactions on*, vol. 15, no. 1, pp. 85–95, 1999.

[14] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," in *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, vol. 2, pp. 1327–1332, 2002.

[15] A. Das, R. Fierro, V. Kumar, J. Ostrowski, J. Spletzer, and C. Taylor, "A vision-based formation control framework," *Robotics and Automation, IEEE Transactions on*, vol. 18, no. 5, pp. 813–825, 2002.

[16] M. J. Mataric, "Interaction and intelligent behavior," tech. rep., 1994.

[17] F. Mondada, E. Franzi, and A. Guignard, "The Development of Khepera," in *Experiments with the Mini-Robot Khepera, Proceedings of the First International Khepera Workshop*, HNI-Verlagsschriftenreihe, Heinz Nixdorf Institut, pp. 7–14, 1999.

[18] J. McLurkin, *Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots.* S.M. thesis, Massachusetts Institute of Technology, 2004.

[19] J. McLurkin, *Analysis and Implementation of Distributed Algorithms for Multi-Robot Systems.* Ph.D. thesis, Massachusetts Institute of Technology, 2008.

[20] W. Nguyen and J. Mills, "Multi-robot control for flexible fixtureless assembly of flexible sheet metal auto body parts," in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 3, pp. 2340 –2345 vol.3, apr 1996.

[21] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," in *Proceedings of the 19th national conference on Innovative applications of artificial intelligence - Volume 2*, pp. 1752–1759, AAAI Press, 2007.

[22] Kuka, "Kuka robotics @ONLINE," June 2010.

[23] Kiva, "Kiva systems @ONLINE," June 2010.

[24] S. Chalup, C. Murch, and M. Quinlan, "Machine learning with aibo robots in the four-legged league of robocup," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, pp. 297 –310, may 2007.

[25] J. Bruce, S. Zickler, M. Licitra, and M. Veloso, "Cmdragons: Dynamic passing and strategy on a champion robot soccer team," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 4074 –4079, may 2008.

[26] S. Kalyanakrishnan and P. Stone, "Learning complementary multiagent behaviors: A case study," in *Proceedings of the RoboCup International Symposium 2009*, Springer Verlag, 2009.

[27] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes, "Coordination for multi-robot exploration and mapping," *Proceedings of the AAAI National Conference on Artificial Intelligence*, 2000.

[28] M. Hsieh, A. Cowley, V. Kumar, and C. Taylor, "Towards the deployment of a mobile robot network with end-to-end performance guarantees," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 2085–2090, 2006.

[29] W. M. Spears, D. F. Spears, J. C. Hamann, and R. Heil, "Distributed, Physics-Based control of swarms of vehicles," *Autonomous Robots*, vol. 17, no. 2, pp. 137–162, 2004.

[30] H. Kitano, M. Tambe, P. Stone, M. M. Veloso, S. Coradeschi, E. Osawa, H. Matsubara, I. Noda, and M. Asada, "The robocup synthetic agent challenge 97," in *RoboCup-97: Robot Soccer World Cup I*, (London, UK), pp. 62–73, Springer-Verlag, 1998.

[31] P. Ranganathan, R. Morton, A. Richardson, J. Strom, R. Goeddel, M. Bulic, and E. Olson, "Coordinating a team of robots for urban reconnaisance," in *Proceedings of the Land Warfare Conference (LWC)*, November 2010.

[32] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. L. 0002, B. Marnier, J. Serre, and B. Maisonnier, "The nao humanoid: a combination of performance and affordability," *CoRR*, vol. abs/0807.3223, 2008.

[33] B. Hendrickson, "The molecule problem: Exploiting structure in global optimization," *SIAM Journal on Optimization*, vol. 5, no. 4, pp. 835–857, 1995.

[34] T. Eren, D. Goldenberg, W. Whiteley, Y. Yang, A. Morse, B. Anderson, and P. Belhumeur, "Rigidity, computation, and randomization in network localization," in *Proc. 23rd IEEE Conference on Computer Communications*, vol. 4, pp. 2673–2684, 2004.

[35] R. Nagpal, H. Shrobe, and J. Bachrach, "Organizing a global coordinate system from local information on an ad hoc sensor network," *Proc. of Information Processing in Sensor Networks (IPSN)*, 2003.

[36] S. Capkun, M. Hamdi, and J. Hubaux, "GPS-Free positioning in mobile ad-hoc networks," in *Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 9 - Volume 9*, p. 9008, IEEE Computer Society, 2001.

[37] P. Ranganathan, R. Morton, A. Richardson, J. Strom, R. Goeddel, M. Bulic, and E. Olson, "Coordinating a team of robots for urban reconnaisance," in *Proceedings of the Land Warfare Conference (LWC)*, November 2010.

[38] D. Moore, J. Leonard, D. Rus, and S. Teller, "Robust distributed network localization with noisy range measurements," in *In Proc. 2nd international conference on Embedded networked sensor systems*, pp. 50–61, 2004.

[39] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan, "The cricket location-support system," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, (Boston, Massachusetts, United States), pp. 32–43, ACM, 2000.

[40] B. Sundaram, M. Palaniswami, S. Reddy, and M. Sinickas, "Radar localization with multiple unmanned aerial vehicles using support vector regression," in *Intelligent Sensing and Information Processing, 2005. ICISIP 2005. Third International Conference on*, pp. 232 –237, 2005.

[41] L. Montesano, J. Gaspar, J. Santos-Victor, and L. Montano, "Cooperative localization by fusing vision-based bearing measurements and motion," in *Intelligent Robots and*

Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on, pp. 2333 – 2338, Aug. 2005.

[42] S. Loizou and V. Kumar, "Biologically inspired bearing-only navigation and tracking," in *Decision and Control, 2007 46th IEEE Conference on*, pp. 1386 –1391, 2007.

[43] R. Ghrist, D. Lipsky, S. Poduri, and G. S. Sukhatme, "Surrounding nodes in coordinate-free networks," in *Workshop on the Algorithmic Foundations of Robotics*, 2006.

[44] N. J. Nilsson, "Shakey the robot," Tech. Rep. 323, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Apr 1984.

[45] H. Moravec, "The stanford cart and the cmu rover," *Proceedings of the IEEE*, vol. 71, pp. 872 – 884, july 1983.

[46] L. Erickson, J. Knuth, J. O'Kane, and S. LaValle, "Probabilistic localization with a blind robot," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 1821 –1827, may 2008.

[47] J. Pugh, X. Raemy, C. Favre, R. Falconi, and A. Martinoli, "A fast onboard relative positioning module for multirobot systems," *Mechatronics, IEEE/ASME Transactions on*, vol. 14, pp. 151 –162, april 2009.

[48] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli, "The e-puck, a Robot Designed for Education in Engineering," in *Proceedingsofthe9thConferenceonAutonomousRobotSystemsandCompetitions*, vol. 1, pp. 59–65, IPCB:InstitutoPolitcnicodeCasteloBranco, 2009.

[49] N. Michael, J. Fink, and V. Kumar, "Experimental testbed for large multirobot teams," *Robotics Automation Magazine, IEEE*, vol. 15, pp. 53 –61, march 2008.

[50] C.-S. Park, D. Kim, B.-J. You, and S.-R. Oh, "Characterization of the hokuyo ubg-04lx-f01 2d laser rangefinder," in *RO-MAN, 2010 IEEE*, pp. 385 –390, sept. 2010.

[51] K. Konolige, J. Augenbraun, N. Donaldson, C. Fiebig, and P. Shah, "A low-cost laser distance sensor," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 3002 –3008, may 2008.

[52] J. Rykowski, *Pose Estimation With Low-Resolution Bearing-Only Sensors.* M.S. thesis, Rice University, 2010.

[53] N. Abramson, "The aloha system: Another alternative for computer communications," Technical Report B70-1, University of Hawaii, Honolulu, Hawaii, Apr. 1970.

[54] J. Feddema, C. Lewis, and D. Schoenwald, "Decentralized control of cooperative robotic vehicles: theory and application," *Robotics and Automation, IEEE Transactions on*, vol. 18, no. 5, pp. 852–864, 2002.

[55] N. Michael, J. Fink, and V. Kumar, "Experimental testbed for large multirobot teams," *Robotics & Automation Magazine, IEEE*, vol. 15, no. 1, pp. 53–61, 2008.

[56] M. J. Mataric, *Interaction and Intelligent Behavior.* PhD thesis, Massachusetts Institute of Technology, 1994.

[57] M. Veloso, M. Bowling, S. Achim, K. Han, and P. Stone, "The cmunited-98 champion small-robot team," in *RoboCup-98: Robot Soccer World Cup II*, vol. 1604 of *Lecture Notes in Computer Science*, pp. 77–92, ACM, 1998.

[58] T. Lochmatter, P. Roduit, C. Cianci, N. Correll, J. Jacot, and A. Martinoli, "SwisTrack - A Flexible Open Source Tracking Software for Multi-Agent Systems," in *Proceedings of the IEEE/RSJ 2008 International Conference on Intelligent Robots and Systems (IROS 2008)*, pp. 4004–4010, IEEE, 2008.

[59] E. Olson, "Apriltag: A robust and flexible multi-purpose fiducial system," tech. rep., University of Michigan APRIL Laboratory, May 2010.

[60] J. McLurkin, "Experiment design for large Multi-Robot systems," in *Robotics: Science and Systems, Workshop on Good Experimental Methodology in Robotics*, (Seattle, WA, USA), June 2009.

[61] A. Das, R. Fierro, V. Kumar, J. Ostrowski, and C. J. Taylor, "A vision-based formation control framework," *IEEE Transactions on Robotics and Automation*, vol. 18, pp. 813–826, October 2002.

[62] A. Cornejo, M. Khabbazian, and J. McLurkin, "Distributed localization for multi-robot systems with large populations of simple robots," *Submitted for Publication*, 2011.

[63] A. Basu, J. Gao, J. Mitchell, and G. Sabhnani, "Distributed localization using noisy distance and angle information," in *Proc. 7th ACM international symposium on Mobile ad hoc networking and computing*, pp. 262–273, 2006.

[64] K. Dogancay, "Bearings-only target localization using total least squares," *Signal processing*, vol. 85, no. 9, pp. 1695–1710, 2005.

[65] D. Niculescu and B. Nath, "Ad hoc positioning system (APS) using AOA," in *Proc. 22nd IEEE Computer and Communications*, vol. 3, pp. 1734–1743, 2003.

[66] K. E. Bekris, A. A. Argyros, and L. E. Kavraki, "Angle-based methods for mobile robot navigation: Reaching the entire plane," in *IEEE International Conference on Robotics and Automation (ICRA04)*, (New Orleans, LA), April 2004 2004.

[67] K. E. Bekris, *Reactive Range-Free Landmark Navigation without Scene Reconstruction.* PhD thesis, Rice University (MS Thesis), Houston, TX, 02/2004 2004.

[68] A. Howard, L. E. Parker, and G. S. Sukhatme, "Experiments with large heterogeneous mobile robot team: Exploration, mapping, deployment and detection," *International Journal of Robotics Research*, vol. 25, pp. 431–447, May 2006.

[69] J. Pugh and A. Martinoli, "Relative localization and communication module for small-scale multi-robot systems," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 188–193, 2006.

[70] A. Gutierrez, A. Campo, M. Dorigo, J. Donate, F. Monasterio-Huelin, and L. Magdalena, "Open e-puck range and bearing miniaturized board for local communication in swarm robotics," in *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, (Kobe, Japan), pp. 1745–1750, IEEE Press, 2009.

[71] W. Whiteley, "Matroids from Discrete Geometry," *AMS Contemporary Mathematics*, vol. 197, pp. 171–312, 1996.

[72] J. Bruck, J. Gao, and A. A. Jiang, "Localization and routing in sensor networks by local angle information," *ACM Transactions on Sensor Networks*, vol. 5, pp. 7:1–7:11, Feb. 2009.

[73] J. D. Horton, "A Polynomial-Time algorithm to find the shortest cycle basis of a graph," *SIAM Journal on Computing*, vol. 16, no. 2, p. 358, 1987.

[74] A. Cornejo, A. J. Lynch, E. Fudge, S. Bielstein, and J. McLurkin, "A practical implementation of scale-free coordinates for multi-robot systems with bearing-only sensors," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, may 2012.

[75] B. R. Donald, "On information invariants in robotics," *Artificial Intelligence*, vol. 72, no. 1-2, pp. 217–304, 1995.

[76] M. Erdmann, "Towards Task-Level planning: Action-Based sensor design," Tech. Rep. CMU-RI-TR-92-03, Pittsburgh, PA, Feb. 1992.

[77] J. M. O'Kane and S. M. LaValle, "On comparing the power of mobile robots," *Proceedings of Robotics: Science and Systems*, 2006.

[78] L. Kleinrock and J. Silvester, "Optimum transmission radii for packet radio networks or why six is a magic number," *Proceedings of the IEEE National Telecommunications Conference*, vol. 4, pp. 1–5, 1978.

[79] J. Yu, S. LaValle, and D. Liberzon, "Rendezvous without coordinates," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pp. 1803 –1808, dec. 2008.

[80] M. Schwager, J. McLurkin, J. J. E. Slotine, and D. Rus, "From theory to practice: Distributed coverage control experiments with groups of robots," in *Proceedings of International Symposium on Experimental Robotics*, (Athens, Greece), 2008.

Appendix A

Appendix

This appendix is a design summary of the electronics and software included on the r-one. The section goes into more detail about the decisions and performance of the r-one system.

A.1 Electrical Design

The r-one electronics was designed using Altium printed circuit board (PCB) design software. This section will overview the schematics and functional description of each system. The design period for the r-one began in summer 2010 and ended with version 6 (V6) shown in Figure A.1(a). Many prototype barebone circuit board versions were tested leading up to the final design as shown in Figure A.1(b). Another design session began in summer 2011 which concluded at version 11 (V11) shown in Figure A.1(c). The final version of the circuit board has a black soldermask and white silkscreen.

(a) r-one V6. (b) circuit boards. (c) r-one V11.

Figure A.1: **a:** The r-one robot V6 finished in summer 2010. **b:** The r-one robot V11 finished in summer 2011.

The goal in the original design was to have a single microcontroller to handle all sensors and device interfaces. However between the V6 and V11 version, an additional microcontroller was added to the design to handle the additional bump sensors.

A.1.1 Architecture

The r-one features a 32-bit Texas Instruments Stellaris LM3S8962 ARM Cortex-M3 microcontroller running at 50 mhz with 64 KB of SRAM and 256 KB flash storage. The Stellaris 8962 is designed to handle the sensors and control the state of the robot. The V6 has the Stellaris 8962 handle all operations in the robot. In the V11 version, an additional *MSP430*F2132 16-bit microcontroller was added to handle bump sensors and additional sensors. The MSP430 includes 8K Flash and 512 Bytes of RAM and enables the robot to enter low-power sleep mode.

The MSP430 is on the bottom circuit board and interfaces with the 8962 microcontroller through SPI as shown in Figure A.2- A.3. The schematics for the two microcontrollers are shown in Figures A.4- A.5.

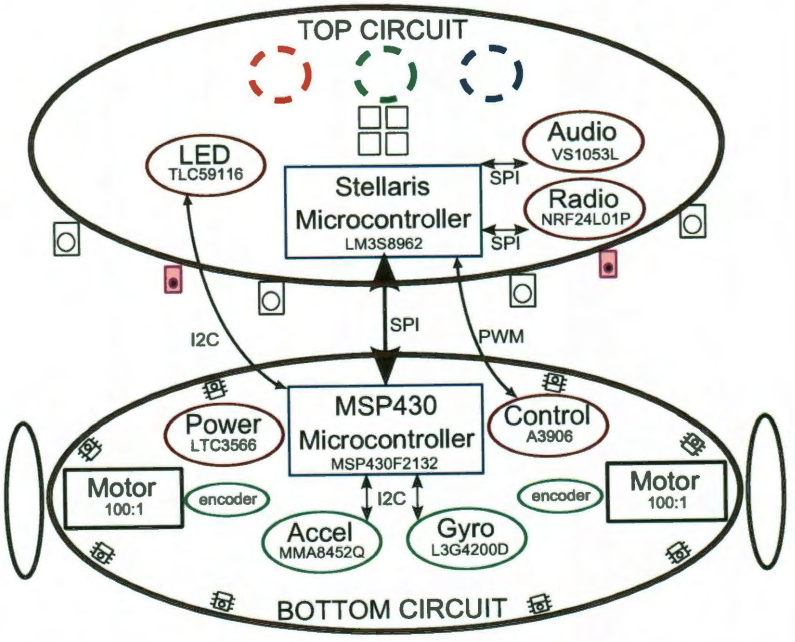


Figure A.2: A system block diagram of devices on the r-one robot. The top and bottom circuit boards feature microcontrollers in blue, sensors in green, interface devices in red.

A.1.2 Bump Sensors

Another critical sub-system to the r-one is the bump sensors. The bump sensors use infrared reflection to measure changes in the free-floating bump skirt on the robot. Figure A.6 illustrates the electrical schematic of these sensors. Infrared reflection sensing can draw tremendous amounts of power if not carefully designed. This design spent many iterations searching for a balance between power and functionality to ensure the bump skirt triggered effectively.

A.1.3 Audio and LED Feedback

Getting feedback from the robot's state is a complicated problem with hundreds of robots. Using a standard LCD display on the robot is not ideal because the user needs to simultaneously view dozens of robots at a time. Therefore LED lights are of the most direct indicator of robot state. The r-one features 15 independently controlled LEDs divided into five red, green and blue groups. The MSP430 controls the I2C communication between the Texas Instruments LED driver (TLC59116F) which powers each LED as shown in Figure A.7.

In the V6 to V11 change in the r-one design, audio feedback was added for additional means of relaying information to the user. The audio schematic uses a MIDI based VS1053 chipset from VLSI solutions. The 8962 microcontroller sends SPI packets to the audio chipset and a series of analog amplification stages boost the signal to a internally mounted speaker on the robot. The audio schematic shown in Figure A.8 required significant electrical testing to achieve the proper gain for pleasant audio feedback.

A.1.4 Motor Encoder System

The motor encoder system is directly mounted on the bottom circuit board. The 8962 Stellaris microcontroller controls the two motors with pulse-width modulation (PWM) to the Allegro A3906 motor controller. The Stellaris also reads the custom built infrared interruption quadrature encoder circuitry. Figure A.9 illustrates the motor and encoder system on the bottom circuit board.

A.1.5 Power Management

The power system regulates $3.3volts$ to the robot from the $3.7volt$ Lithium polymer battery. The power system also charges the battery from a $5volt$ USB connector. An additional mode of operation is the power sleep mode controlled by the MSP430. Power sleep allows the robot to remain active while drawing micro-amps of power. Figure A.10 illustrates the Linear power regulator electrical schematic.

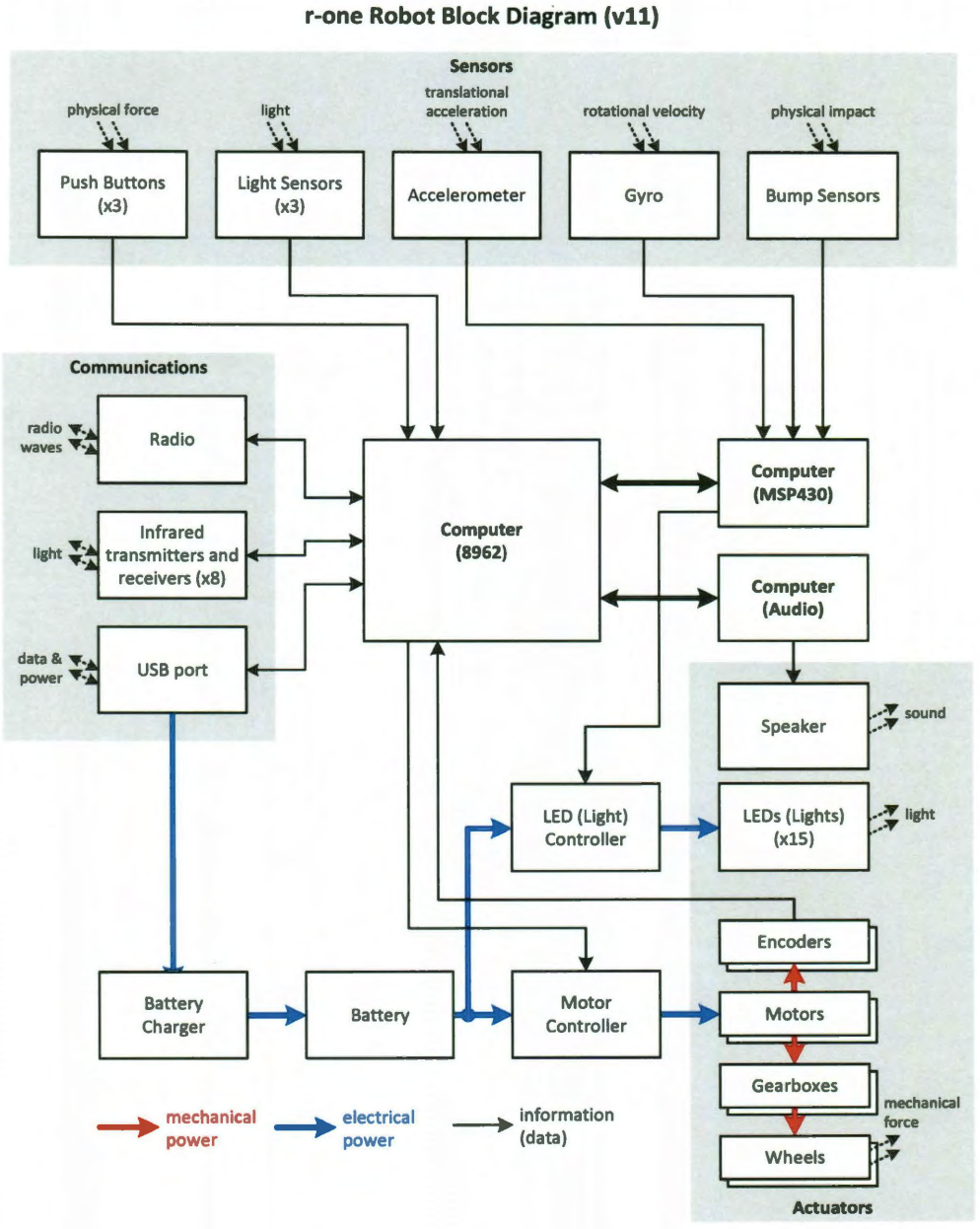


Figure A.3: A complete block diagram of systems on the r-one robot. The two circuit boards are abstracted in this diagram to focus on the power and information layers between the robot. The diagram shows mechanical power in red, electrical power in blue and information in black.

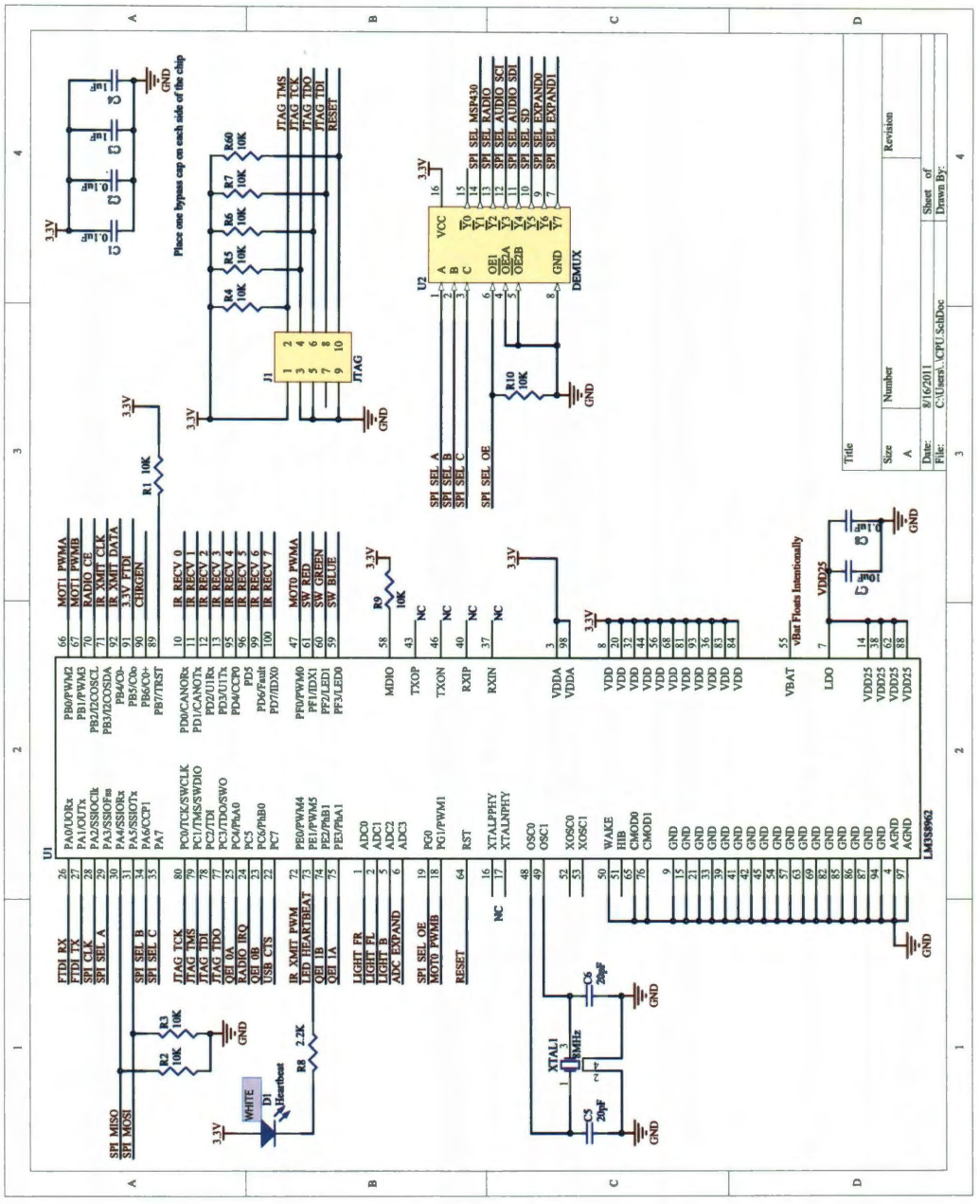


Figure A.4: The schematic of the 8962 microcontroller. A 10-pin JTAG connector is mounted on the robot to program. A demutiplexer is also mounted to obtain additional select lines for SPI communication.

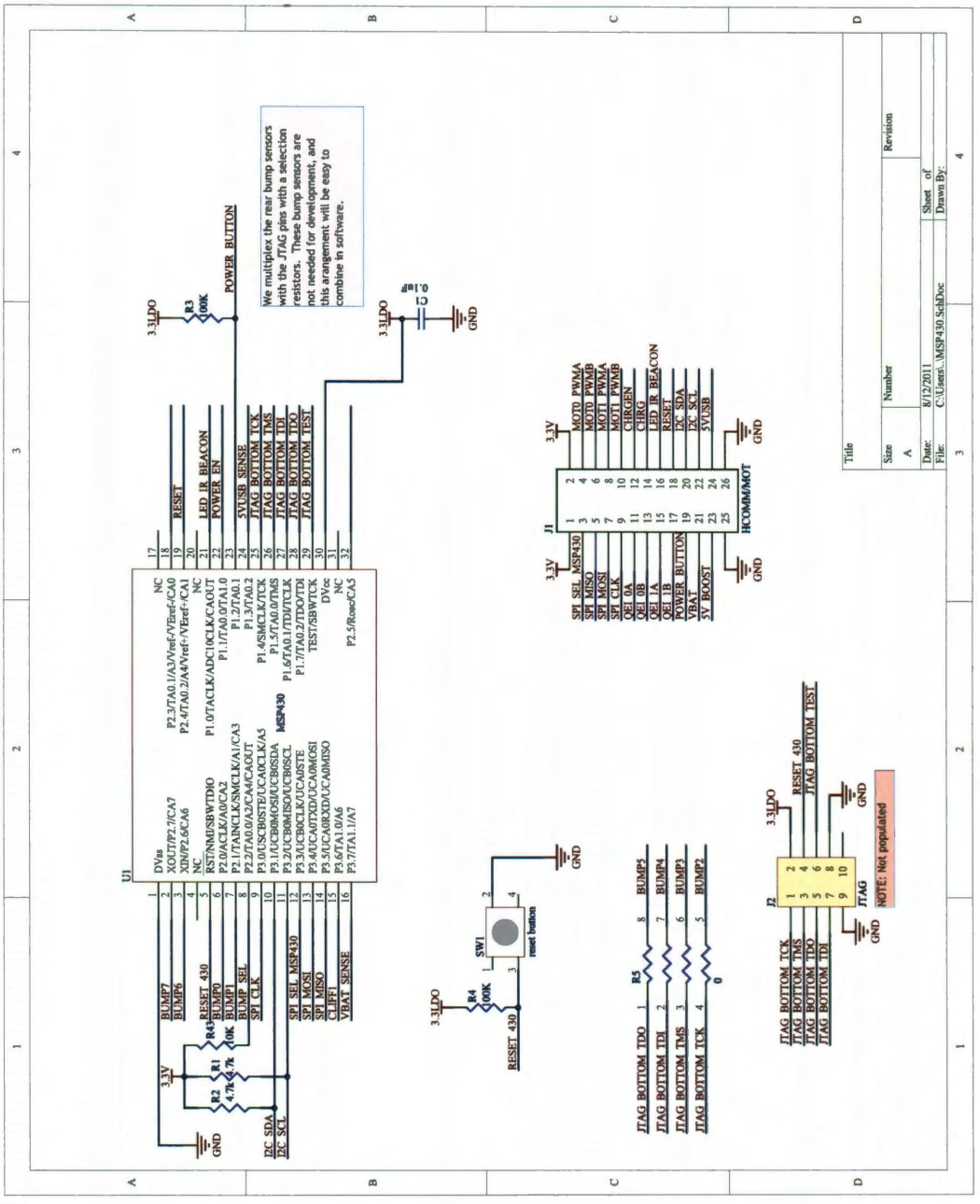


Figure A.5: The schematic of the *MSP430*F2132 micro-controller on the r-one bottom circuit board. The MSP430 connects to bump sensors directly and SPI to the Stellaris 8962. The I2C interface is also used to communicate to an accelerometer, gryo and LED driver located on the top board. A JTAG connection is not populated but available on the bottom board to program the MSP430. For ease of use, a special adapter is used to JTAG program the bottom board from underneath without mounting a connector.

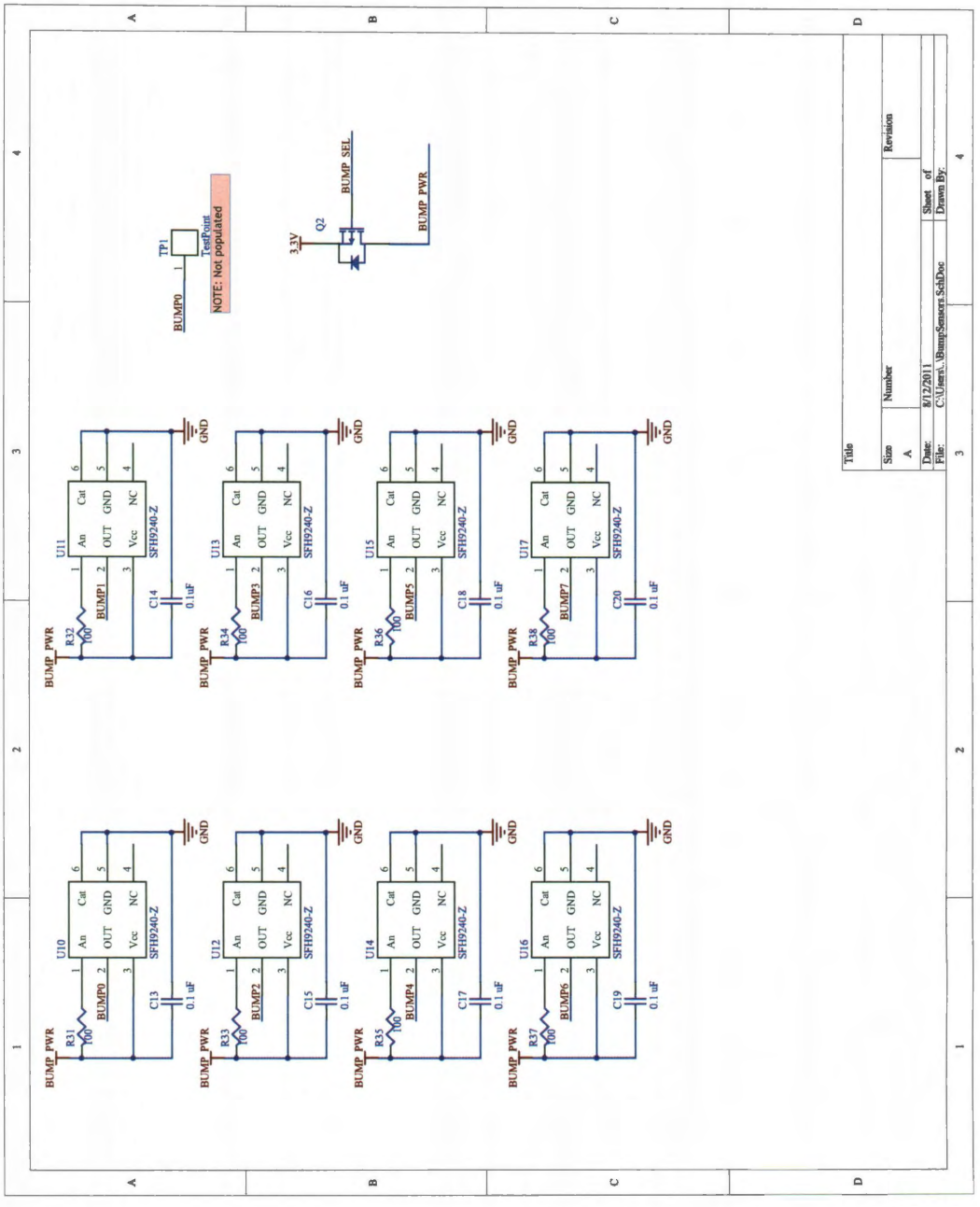


Figure A.6: The schematic of the infrared reflection bump sensors (SFH9240-Z) connected to the MSP430.

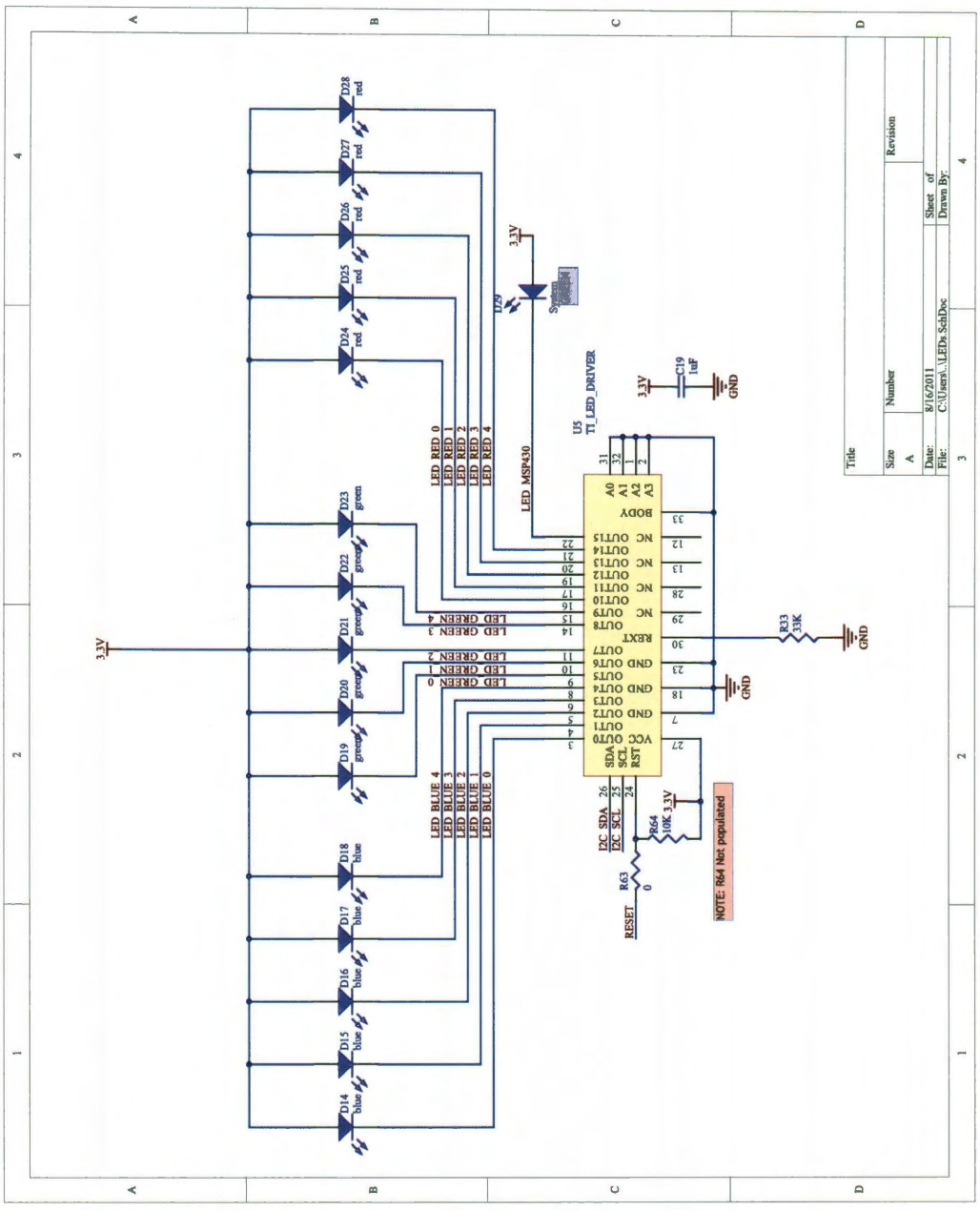


Figure A.7: The schematic of the LED driver (TLC59116F) connected on the top circuit board.

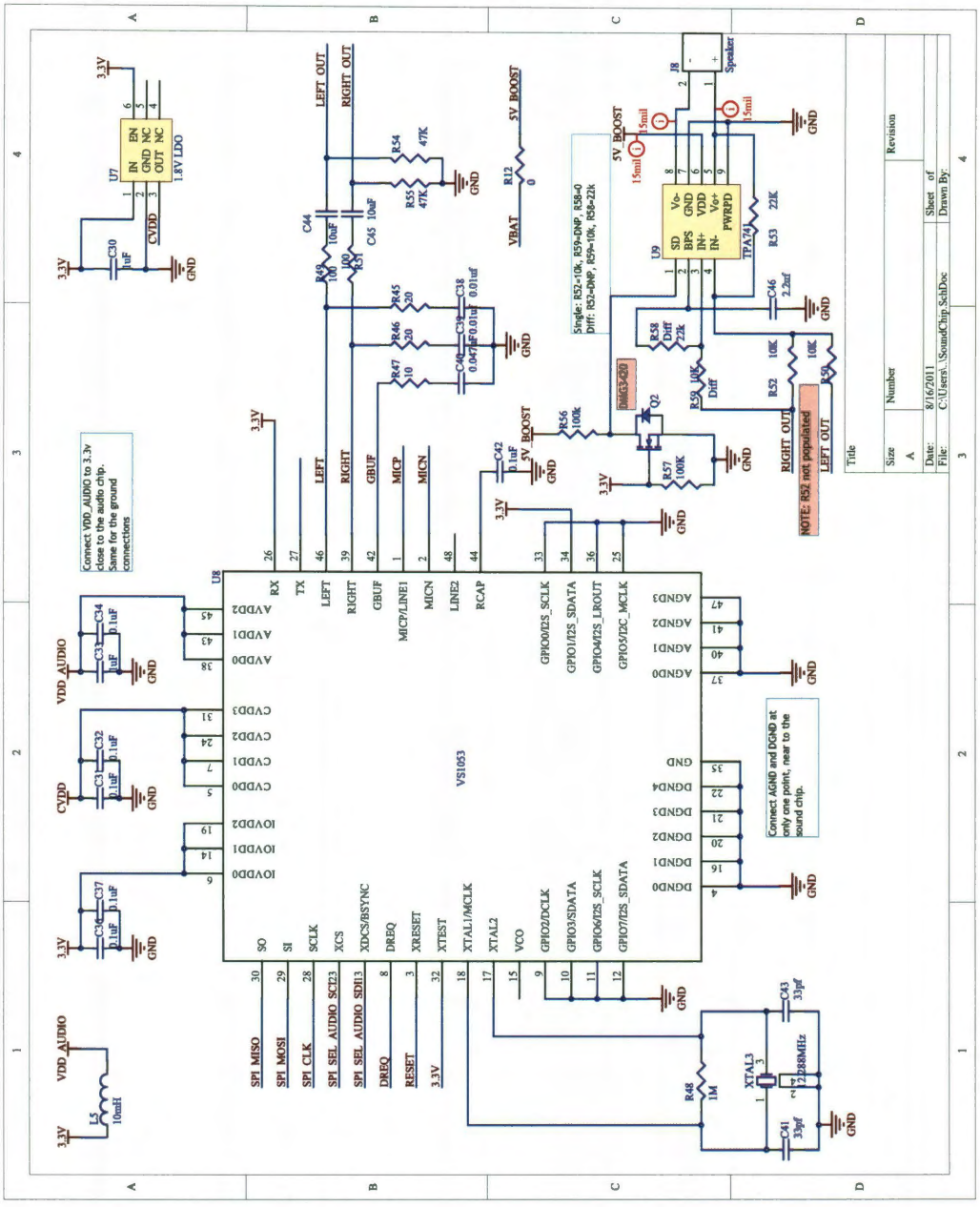


Figure A.8: The schematic of the Audio driver (VS1053) connected on the top circuit board.

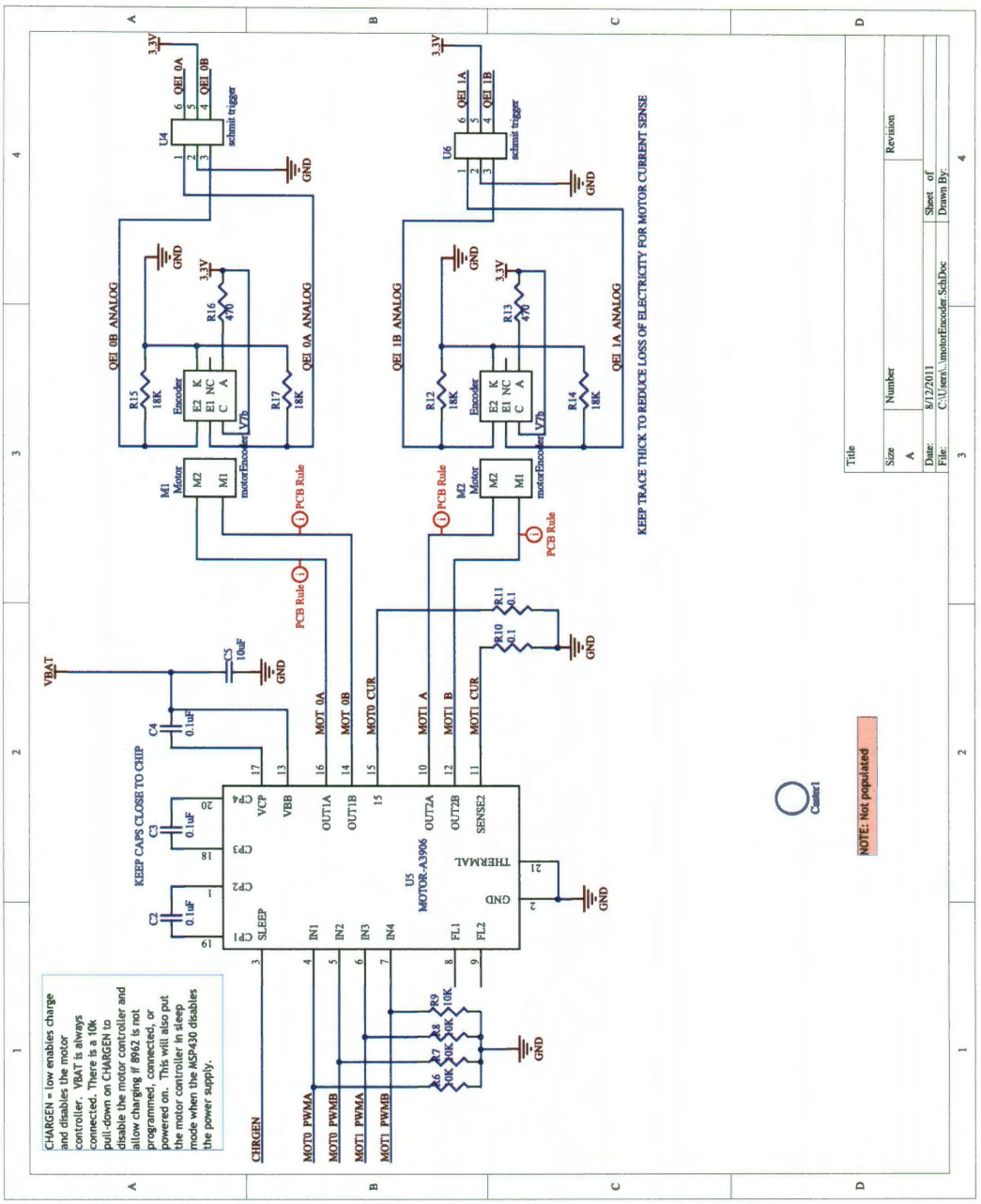


Figure A.9: The schematic of the motor driver (A3906) and the infrared interruption encoder circuitry.

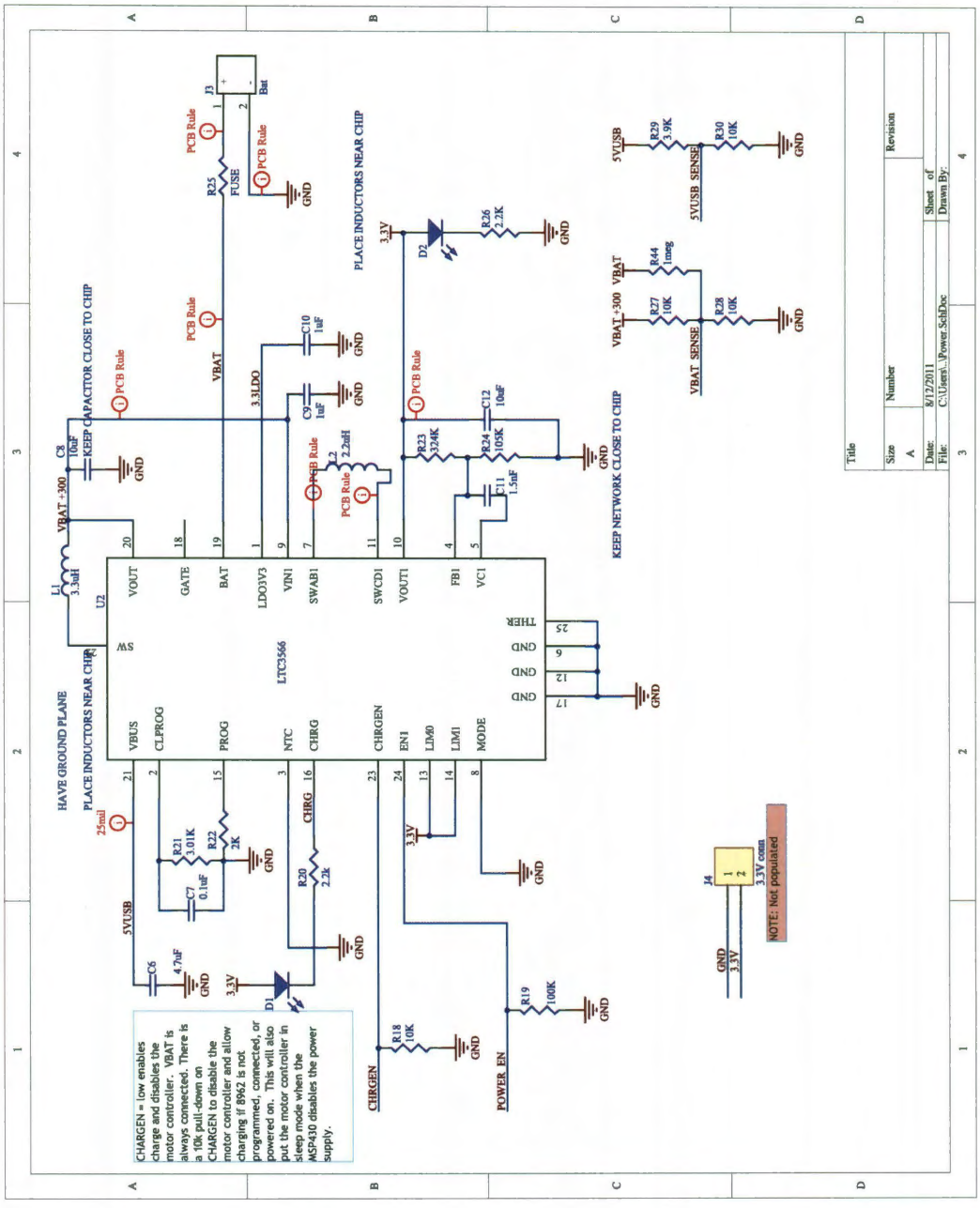


Figure A.10: The schematic of the power controller (LTC3566).

A.1.6 Radio Communication

Radio communication over 2.4Ghz radio is primarily focused on data collection and not intended for use in distributed algorithms. Many types of radios were tested before settling on the Nordic radio chipset. Our testing found the screw-on antenna 1/4 wavelength provides reliable at least 50 feet range of communication. Further radio distances were not thoroughly investigated for these robots since 50 feet is sufficient for most indoor applications. An embedded antenna was also tested with less successful results. Figure A.11 illustrates the radio schematic.

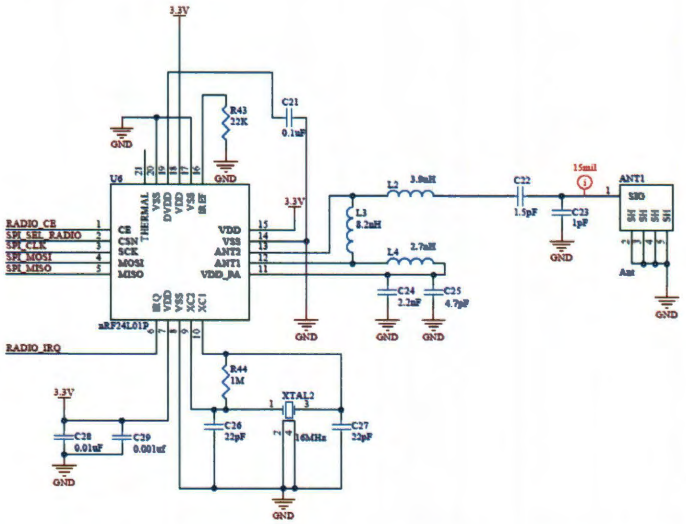


Figure A.11: The schematic of the Nordic nRF24L01P radio connected to the 8962 over SPI.

A.1.7 Infrared Communication

The key subsystem on the r-one that separates it from other robot systems is the distributed communication over infrared. Each robot has eight IR transmitters and eight receivers. The receivers are Sharp IR remote control devices with 38khz modulation and a maximum bit rate of 1250bps. The transmitters a IR LEDs that transmit an encoded message with the robot ID and relevant geometric information between neighbors. The receivers do not measure the power and hence do not measure range of the receivers. The Sharp receivers used are low-cost and do not provide analog measurements for range. Measuring only bearing and orientation between robots was a pivotal design choice on the robot to reduce cost and complexity. Figure A.12 demonstrates the simplicity of the connections to each infrared system to the Stellaris 8962 microcontroller.

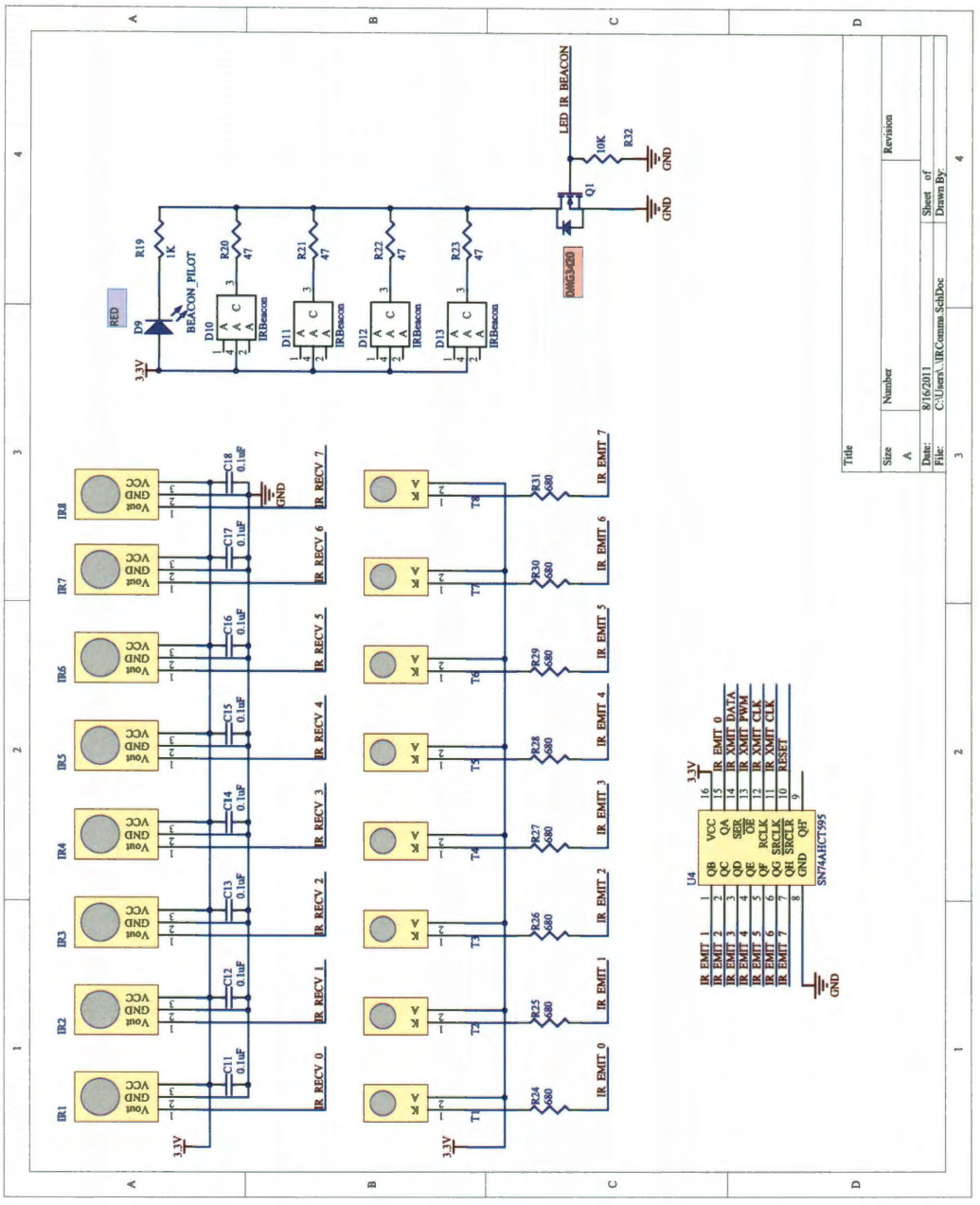
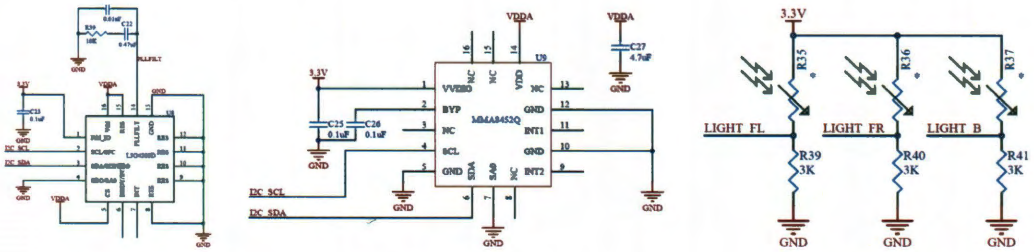


Figure A.12: The schematic of the infrared receivers and emitters on the r-one. The Sharp IR receivers directly connect to the Stellaris 8962 and the emitters connect to SN74AHCT595 buffer which sends out an encoded message to the robots. The IRbeacon circuitry is composed of four infrared beacon LEDs mounted in the center of the robot to track the robot position from an overhead camera.



(a) gyro. (b) accel. (c) light sensor.

Figure A.13: **a:** The gyroscope sensor (L3G4200D) on the bottom circuit board. **b:** The accelerometer sensor (MMA8452Q) on the bottom circuit board. **c:** The light sensor mounted on the top circuit board.

A.1.8 Additional Sensors

The r-one also features a gyro, accelerometer, light sensors and a cliff detection sensor. The 3-axis gyro and 3-axis accelerometer communicate with the MSP430 over I2C as shown in Figure A.13(a)- A.13(b). The cliff detection sensor is similar to a bump obstacle sensor but the development is still ongoing with the results of this sensor. Three light sensors or photo-resistors are mounted on the top circuit board as shown in Figure A.13(c).

A.2 Software Design

The r-one supports two different programming languages, python and c. The C implementation runs a variant of Free-RTOS; Real-Time Operating System modified specifically for the r-one. The python implementation uses a framework developed by Rixner and Barr which supports Stellaris based microcontrollers with Python. Python is not discussed at length in this work, however a full tutorial to using Python on the r-one is documented on the Rice University *Introduction to Engineering* (ENGI 128) course website.

A.2.1 roneos

The C programming language on the r-one is based around libraries provided by Texas Instruments and Free-RTOS. A basic c program can be executed without free-RTOS but the operating system provides the foundation blocks to multi-task with multiple sensors. The primary building block will be referred to as the *roneos* library. The three main components are roneLib, driverlib and roneos as shown in Figure A.14. The Multi-robot systems Lab at Rice University has included drivers, device specific tasks and a series of initialization routines in roneos. As an example, a user can quickly utilize roneos to perform advanced behaviors with multiple sensors running simultaneously.

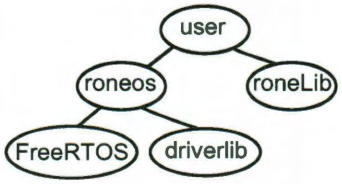


Figure A.14: The software hierarchy a user program will call when using roneos and roneLib.

A.2.2 Device Drivers

Each system detailed in the electrical design has a corresponding device driver to read the sensor directly or communicate with the system. Figure A.15 shows the various levels of communication from the Stellaris 8962 microcontroller. The microntroller directly measures sensors or uses serial peripheral interface (SPI) to communicate with the MSP430, radio or audio. A bus interface between these devices allows for fast exchange of information at a low number of physical general purpose input-output (GPIO). The MSP430 relies on I2C (another type of bus protocol) to communicate to the gyro, accelerometer and LED devices.

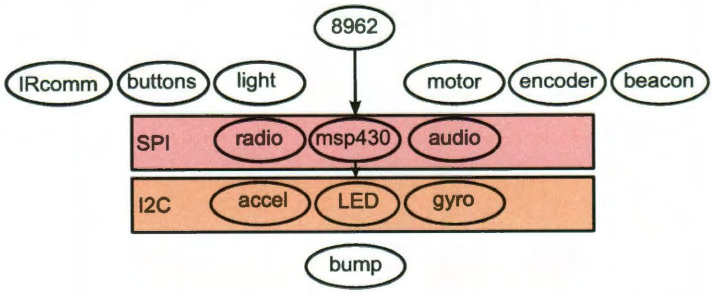


Figure A.15: The available device drivers commanded by the 8962 microcontroller. The MSP430 interfaces over SPI to provide information from bump, gyro and accelerometer.

A.2.3 Threads and roneLib

The general procedure for starting up the r-one is executed from *system.c*. The roneos system initializes the heartbeat LED and unique ID in *systemPreInit*. The robot successfully passes through *systemPreInit* when the white heartbeat LED blinks three times. The triple blink heartbeat is a key indicator that the code is starting successfully.

After the pre-initialization phase has executed, the *systemInit* function initializes all remaining device drivers. The serial communication is given a binary semaphore to allow for safe shared access. Serial communication is how the robot outputs information over the USB port. The command *cprintf* allows the user to print readable messages in C-like printf format. However, cprintf only supports a limited subset of printf. The cprint function uses a semaphore mutex to access the printing utilities.

The SPI bus also uses a semaphore mutex to share SPI access between the MSP430, radio and audio devices. The radio initialization sets up a receive and transmit queue to handle data traffic. If the robot receives an incoming wireless packet, the interrupt handler triggers and fills the radio queue. Audio is the simplest of the three SPI devices because the user only sends information to the audio device.

The MSP430 micocontroller located on the bottom circuit board controls the gyro, accelerometer, LED driver and bump sensors. The I2C bus is controlled by the MSP430, therefore roneos can only read the values of the devices on I2C through the MSP430 SPI communication. The SPI bus is the main source of contention on the r-one due to data traffic shared between three devices. In addition to the MSP430 SPI interrupt, the r-one also has interrupts on the quadrature encoders and IR communication. The IR communication is interrupted when new data is received on the IR receivers and then fills a queue with the corresponding data. Figure A.16 shows the a sample illustration of tasks and interrupts running on roneos. The user has full control of the behavior and background tasks for performing experiments with the r-one.

A.2.4 Sample Program

The r-one C code is developed around Sourcery CodeBench (formerly Sourcery G++) to compile roneos, roneLib and the user project code. The framework is similar to Eclipse and provides an excellent debugging utility to inspect variables while the robot is in operation.

A sample start-up program will run *systemPreInit*(), *systemInit*() and a task called *neighborsTask* by calling $neighborsInit(600, FALSE)$. The neighbors task runs at a $600ms$ period and collects neighbor information from the IR communication queue. Once the system is initialized, the user can declare a task in the operating system:

$$osTaskCreate(behaviorTask, behavior, 4096, 0, Priority)$$

The *behaviorTask* represents the a user robot behavior such as avoid obstacles or drive straight. Additional tasks can also be created such as a background task, refer to the

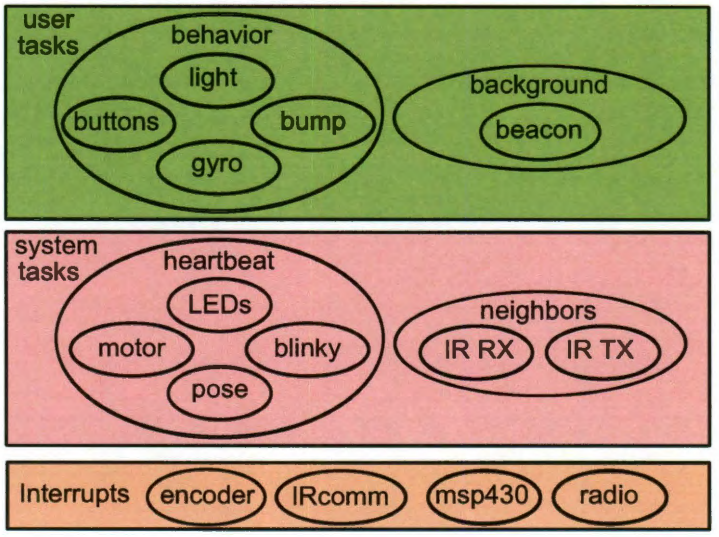


Figure A.16: A sample illustration of the tasks and interrupts running on roneos. The interrupts are built into the operating system and the system tasks also are configured to run at a regular update rate. The heartbeat task is the most critical task which updates the LEDs, blinky heartbeat, pose control or direct motor commands at 10 hertz. The second system task typically run consists of updating the neighbor communication over IR. This task runs every $600ms$. Each one of these tasks relies on interrupts to obtain information at the rate of the sensors update. Finally, the user tasks are given as a sample illustration of running a behavior or background tasks. The user has the ability to use any type of sensor in the behavior thread. The background task is typically for lower update rate operations like updating the beacon or computation.

example projects for more detail. Once the tasks are created, the operating system starts all tasks by calling:

$$osTaskStartScheduler()$$

A sample library called roneLib has been developed for a common set of r-one software routines with movement behaviors, radio data structures and neighbor list operations. A user program will save time using the roneLib routines and hopefully provides guidelines for handling neighbor communication. Future development on the r-one will consist of adding many common behaviors to roneLib such as localization and obstacle mapping.

Once your workspace and repository is properly configured, the user can view BigFat-StinkingDemo (BFSD) project as a walk-through of running a program on the r-one. The BFSD project demonstrates multiple behaviors such as multi-robot flocking with obstacle avoidance.

A.3 Data Collection

Collecting ground-truth or global information from the robots is an important step to monitor the state of distributed algorithms. The r-one supports two different data collection schemes with APRIL tags and another using beacon tracking. The two methods both use cameras to track the positions of the robots in the workspace. The data collection system is interchangeable between both tracking methods. The system consists of a camera, host computer and a camera computer as shown in Figure A.17.

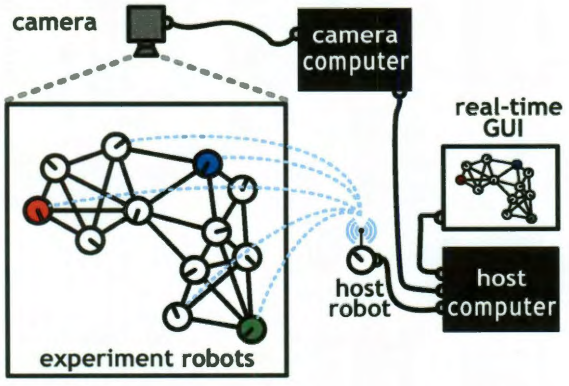
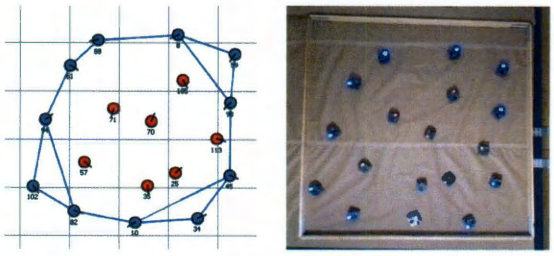


Figure A.17: Diagram of the r-one data collection system. The ceiling-mounted camera identifies each robot and tracks the $\{ x, y, \theta \}$ positions of each robot simultaneously, reporting the results to the computer at 5 Hz. The camera is interchangeable between the IR beacon or APRIL tag tracking.



(a) Beacon Camera. (b) Beacon Camera Real-time.

Figure A.18: **a:** A camera view from the Newton camera perspective. 25 robots are pictured. **b:** A real-time GUI correspondence to the camera picture of robot positions, (x, y). The robots in blue are on the boundaries. These pictures are tests with the Swarm-Bot from 2007 and do not represent the r-one beacon tracking.

A.3.1 APRIL Tag Tracking

The APRIL tag tracking system consists of a camera(s) and tracking code. The tracking code runs in Java and is built around Ed Olson's APRIL tag software framework. Each robot has a bar-code tracking tag that covers the majority of the robot. The tracking system provides (x, y, θ) global position information of the robots in the camera reference frame. For the majority of the experiments, two logitech webcams were used simultaneously to collect position information of the robot.

The webcams are connected to one computer that runs a tracking program called *SwingCamTag.java*. Two webcams are mounted $1.13m$ inches apart at approximately $1.37m$ from the ground. The visible workspace of the camera mounting allows for a workspace of approximately $1m$ x $2m$.

Multiple low-cost camera tracking provides a scalable interface to potentially expand the workspace to track hundreds of robots simultaneously. However, implementation details with data aggregation of multiple tracking computers is still untested.

A.3.2 Beacon Tracking

The Beacon tracking system was developed by Newton Labs and tracks the robots' (x, y) positions. The IR camera detects an encoded IR beacon pattern at 10 bits/second. Each robot blinks its unique ID on the IR beacon mounted in the center. The robot uses four IR LEDs in unison to increase the brightness of the blink and subsequently increase the range of detection. Figure A.18(a)- A.18(b) illustrates the correspondence between the physical camera view and the real-time positions of the robots.

In contrast to APRIL tags, the beacon tracking does not track robot heading, (θ). For certain experiments only the (x, y) positions are needed to properly plot the results of an algorithm. The main advantage of IR beacon tracking is the large workspace for

experiments. With one IR camera angle mounted at a corner of a ceiling, the r-one can be tracked within a 7.62 meter distance. This one camera covers the majority of the lab workspace for experimentation which provides a suitable area to track large populations of robots. This is a significant improvement from our APRIL tags workspace size.

A.3.3 Data Collection Software

The data collection software consists of ground-truth tracking software, r-one host communications and data aggregation code. The tracking software takes the positions of the robots and outputs packets over a local Ethernet network. The host computer will connect directly to a host r-one robot to capture additional information such as neighbor information between robots. The host communications between the r-one typically consists of parsing a comma-separated-values (CSV) data stream from a USB connection to the host robot. The final part of the system aggregates together tracking information and r-one communications. The aggregator also has the ability to view the information on a real-time GUI or save the data to a CSV file. Figure A.19 illustrates the connections between the different software modules.

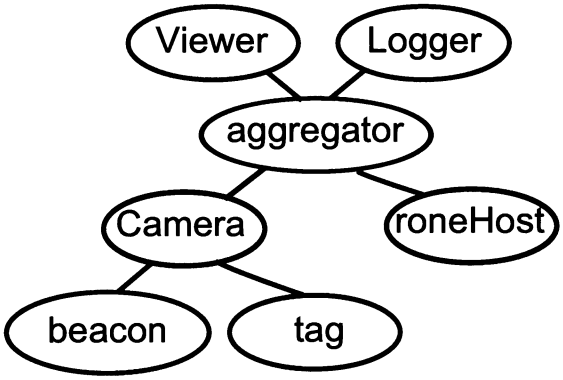


Figure A.19: The hierarchy of the data collection modules. The camera module is configured to collect from APRIL tags or beacons. The camera module sends information to the aggregator through network packets. The roneHost communicates directly to the robot for neighbor information between robots and connects directly to the aggregator program over USB connection. Once the data is aggregated together, the data can be opened in a Viewer module or logged in CSV format in a Logger module.

Each user experiment with the r-one typically has a unique set of information to collect such as scale-free or broadcast hops. To configure this information for data collection, the user would write r-one embedded code to print a CSV data stream. A data aggregation project will also be needed to parse their unique robot CSV data stream. Figure A.20 demonstrates the components of the system that need to be replaced with the unique

information of the experiment. The aggregator merging of data can be modified inside the new user Logger project.

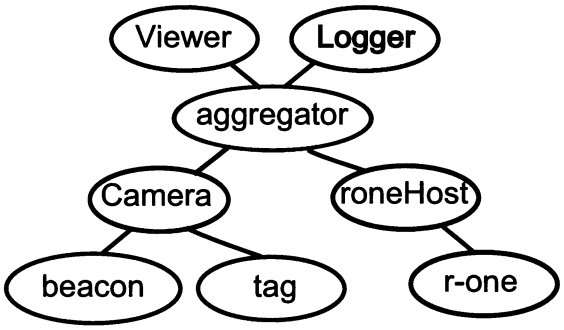


Figure A.20: The hierarchy of the data collection modules with a unique experiment. The unique section of the experiment needs to be changed on the r-one embedded software that communicates to roneHost. If new information is being aggregated together, than the user would write a unique logger program as well.

If only the camera tracking positions are needed in the experiment. A default project called *AprilTagLogger* is configured to save positions of multiple robots using the APRIL tag tracking system. No additional code is required and the process is fairly streamlined.